



How to Design and Develop Smart Solutions

SLIDES: <https://www.snap4city.org/download/video/course/dev/Snap4City-8th-Solutions-Development-v9-6.pdf>

To get updated version of the slides go to <https://www.snap4city.org/577>

Updated interactive SLIDS: <https://www.snap4city.org/download/video/course/dev/>

VIDEO on overview of this document: https://youtu.be/UtaZy_A8qTo

From Snap4City:

- We suggest you read the TECHNICAL OVERVIEW:
 - <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>
- See Client-Side Business Logic Widget
Manual: <https://www.snap4city.org/download/video/ClientSideBusinessLogic-WidgetManual.pdf>
- <https://www.snap4city.org>
- <https://www.snap4solutions.org>
- <https://www.snap4industry.org>
- <https://twitter.com/snap4city>
- <https://www.facebook.com/snap4city>
- <https://www.youtube.com/channel/UC3tAO09EbNba8f2-u4vandg>

Coordinator: Paolo Nesi, Paolo.nesi@unifi.it

DISIT Lab, <https://www.disit.org>

DINFO dept of University of Florence,

Via S. Marta 3, 50139, Firenze, Italy

Phone: +39-335-5668674

Access Level: public

Date: 24-09-2023

Version: 7.7





Summary

I.	Introduction.....	4
I.A.	Registration and trial on Snap4City.org.....	5
I.B.	Change of Terminology, for 2023 version of the platform.....	6
II.	Architectural Flows and Functional Areas	7
III.	Snap4City Architecture	9
III.A.	Considerations and Remarks	18
IV.	Development Life Cycle	20
IV.A.	Analysis Phases.....	22
IV.A.1.	Innovation Matrix by domain, and Entity identification	23
IV.A.2.	The Dictionary via entity identification: physical, virtual and modules/tools.....	25
IV.A.3.	Analysis: Scenarios formalization	29
IV.A.4.	Analysis: Use Cases formalization	33
IV.A.5.	Analysis: Requirements formalization.....	33
IV.A.6.	Analysis: Sequence Diagrams formalization.....	35
IV.B.	Design Phases	36
IV.B.1.	Design: Data Discovery	36
IV.B.2.	Design: Data Modelling	37
IV.B.2.a-	Data: High Level Types, HLT.....	38
IV.B.2.b-	From Data Modeling to Entity Messages	41
IV.B.2.c-	How to perform Data Modeling	42
IV.B.2.d-	Concept of ServiceURI and HLT Identifiers.....	45
IV.B.2.e-	Concept of Time Series	46
IV.B.2.f-	Rule for Entity Models/Instances, names and values.....	48
IV.B.2.g-	Examples of Data Models	50
IV.B.2.h-	How to Create Entity Instances / IoT Devices, and their messages	53
IV.B.2.i-	High Level Type vs Storage and distribution channel.....	54
IV.B.3.	Design of Data Processes, Processing Logic	55
IV.B.3.a-	Thumb Rules on Snap4City Processing Logic development	58
IV.B.3.b-	Example of Processing Logic (IoT App) Design, for each independent Flow	59
IV.B.3.c-	Main Snap4City Proc.Logic / IoT App nodes/block for Node-RED.....	60
IV.B.3.d-	The power of Snap4City Libraries for Proc.Logic / IoT App Node-RED.....	62
IV.B.3.e-	Snap4City Proc.Logic / IoT App with Debug Option	63
IV.B.3.f-	Why and How to use delegations.....	65
IV.B.3.g-	Blockchain support on Snap4City	66
IV.B.3.h-	Snap4City integrated with Milestone X protect VMS.....	67
IV.B.4.	Design of Data Analytics	70
IV.B.5.	Design: User Interface and Business Logic	74
IV.B.5.a-	Passive and Active Dashboards/views.....	74
IV.B.5.b-	Summary of Dashboard Widgets' Capabilities for Business Logics	76
IV.B.5.c-	Design process for graphics user interface.....	77
IV.B.5.d-	Example of Dashboard Schema	78
IV.C.	Development Phases	80

IV.C.1.	Development: Data Processes, aka Processing Logic (IoT Applications).....	80
IV.C.1.a-	Capabilities of Proc.Logic / IoT App in Snap4City	81
IV.C.1.b-	Examples: Processing Logic / IoT App: typical patterns	83
IV.C.1.c-	Typical strange patterns that may be not efficient in most cases, or wrong:	84
IV.C.1.d-	Example of Trigger / Synchronization of Device/Entity Data (Pattern)	85
IV.C.1.e-	Delete Devices	86
IV.C.1.f-	Test and Verify Data Ingestion	86
IV.C.1.g-	Test and Deploy of Proc.Logic / IoT App Node-RED Libraries	87
IV.C.2.	Development: Data Analytics aka Python and/or Rstudio processes	87
IV.C.2.a-	Accessing data from Data Analytics Processes.....	88
IV.C.2.b-	Exploit Data Analytics (Python and/or Rstudio) from Proc.Logic/Node-RED	89
IV.C.2.c-	Export data from storage for different purposes	90
IV.C.3.	Development: User Interface as Dashboard	91
IV.C.3.a-	The List of Dashboard Widgets, kinds of widgets.....	95
IV.C.3.b-	The Dashboard Builder	96
IV.C.4.	Development: Server-Side Business Logic, as Processing Logic	97
IV.C.4.a-	Examples of Proc.Logic Event Driven Business Logic Nodes / Patterns	99
IV.C.4.b-	Dashboard Widgets: gates for Server-Side Business Logic.....	100
IV.C.4.c-	Dashboard Widgets: rendering data/device tables on Dashboards.....	103
IV.C.5.	Development: Client-Side Business Logic.....	104
IV.C.6.	Development: external applications and Data Analytics using the Advanced Smart City API	110
IV.C.6.a-	Authentication to API access, REST Call.....	112
IV.C.6.b-	Example: Access to Orion Broker API	115
IV.C.6.c-	Example: forbidden characters by the Orion Broker.....	116
IV.C.6.d-	Example: Get Data using Smart City API.....	116
IV.C.6.e-	Example of Registering a New User via Mobile App	116
IV.C.6.f-	Example: Get access token/refresh token via user credentials	117
IV.C.6.g-	Example: Get access token via refresh token.....	118
IV.C.6.h-	Example: Using Advanced Smart City API in Python	118
IV.C.6.i-	Snap4City Internal API, REST Calls	120
IV.D.	Testing and Deploy Phases	121
IV.E.	Validation and Production Phases.....	121
V.	Other Aspects	123
V.A.	Authentication and Authorization, integration	123
V.B.	Creation of Smart Applications exploiting the platform	123
V.C.	Nonfunctional aspects of the Snap4City Development Environment.....	123
V.D.	Installation on Premise	123
VI.	Selected References	124
VI.A.	Training Course on Slides and Interactive Slides, Videos	128
VII.	Appendix: Data Dictionary for Entity Models (IoT Device Models) and Entity Instances (Devices), Entity Variables, attributes and metrics	129

I. Introduction

Snap4City based on Explainable Artificial Intelligence provides smart solutions for the city users and decision support systems for operators simplifying and synergically improve the life in city and lands, coping with SDG and 15 MinCityIndexes. Effective ready to use solutions for: mobility and transport, environment, energy, industry, justice, ... Snap4City covers full Digital Twins, from Smart Data Models, High Level Types, to the 3D for the whole city and single building at low costs; enabling the creation of diffuse smart city solutions and Living Lab on cloud and/or on premise, on which stakeholders can easily adopt a plethora of solutions via an almost no coding platform. The Snap4City full stack platform is 100% open source, official from FIWARE, EOSC, Node-RED, etc.

According to scenarios analyzed in developing several smart applications and tools into the multitenant large infrastructure of Snap4City, a best practice has been derived for the development of smart solutions for operators, final users, city users and decision makers. The Smart Solutions are characterized by:

- coping a **large number of data types**, exploiting unified data models and tool, such as: IoT devices (sensors and actuators, may be based on Smart Data Models, IoT Device Models, etc.), POI (Point of Interest), typical time trends, KPI, heatmaps, constrained scenarios, traffic flows, personal data, user profiles, maps, orthomaps, shapes, GIS data, trajectories, origin destination maps, scenarios, etc.; They need to be indexed according to their **semantic relationships** (spatial, geographical and temporal) to facilitate the search and extraction of new knowledge for smart applications;
- **highly dynamic smart city solutions**, from **simple** (data rendering of info and statistic data) to **complex** (early warning, predictions, prescriptions, optimization, what-if analysis and simulations) without limiting the flexibility by relegating logic in coding;
- controlling and managing the **status of the processes** and **data flows** using the same tool and interface. This means that it must be easy to pass: (i) from a data to the processes involved in their production/exploitation, and (ii) from the process/dashboard/application/analytics to the specific data exploited and/or produced, while the API identification is a too generic access;
- **Interoperating** with a large range of third-party applications, which can provide data, retrieve data, intrinsically integrated in the data and process workflow, and perform data analytics, etc.

The above requirements aim at creating a support to be exploited during the whole application life cycle (development, maintenance, exploitation/extension and change) in the dynamic and collaborative environment based on microservices for smart city, and it can be extended in other domains. This would allow the Operators to **optimize the services and (con)training the developers** to exploit data and services rather than reinventing a solution from zero for each new application development. *These requirements motivated the research, and the modelling of the solution described in this paper.* Thus, reducing the time to develop by reducing the time to (i) *identify and exploiting the patterns* developed by other developers, and to (ii) *identify the relationships* to solve the problems. This makes a multi-application framework easy to evolve and scale.

The design and development of these solution is much more complex than classic software developments approaches, and thus the development life cycle has been updated. Peculiar aspects of these distributed and data intensive solutions (which can belong to the cyber-physical solutions) are the need of: (i) fast prototyping, (ii) collaborative activities exploiting and reusing the same component by multiple developers, (iii) integrating data flow and control flow in event driven, (iv) presence data analytic in the development life cycle, (v) front-end tools which request further data analytic and sophisticated business logic, etc. Thus, the Development Life Cycle involves several specialized tools such as those for creating flows and dashboards, etc. For this purpose, in most cases the Snap4City and Snap4Industry developers adopt strongly dynamic microcycles and microservices, exploiting Snap4Tech tools and supporting multiple applications and solutions at the same time, the requirement analysis has to cope with a number of aspects: data ingestion, business logic, data analytics, storage, business intelligence and interactive user interface, also integrating other third-party tools at any level.

We suggest you:

- **TECHNICAL OVERVIEW:**
<https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>
- Client Side Business Logic
<https://www.snap4city.org/download/video/ClientSideBusinessLogic-WidgetManual.pdf>
- Smart City Booklet https://www.snap4city.org/download/video/DPL_SNAP4CITY_2022-v02.pdf
- Industry 4.0 Booklet: https://www.snap4city.org/download/video/DPL_SNAP4INDUSTRY_2022-v03.pdf
- Data Analytic Booklet https://www.snap4city.org/download/video/DPL_SNAP4SOLU.pdf
- Training Course in several parts: <https://www.snap4city.org/drupal/node/577>
- Applicative Scenarios: <https://www.snap4city.org/drupal/node/4>
- **Training videos from Youtube channel:**
<https://www.youtube.com/channel/UC3tAO09EbNba8f2-u4vandg>
- News and technical news on: <https://www.snap4city.org/135>
- Newsletter on social media and on Snap4City:
<https://www.snap4city.org/drupal/taxonomy/term/7>
- Research papers:
 - <https://www.snap4city.org/drupal/node/426>
 - <https://www.disit.org/node/5487>
 - <https://scholar.google.com/citations?user=c2S3Ni0AAAAJ&hl=en>

Dissemination From Snap4City:

- <https://www.snap4solutions.org>
- <https://www.snap4city.org>
- <https://www.snap4industry.org>
- <https://www.linkedin.com/company/snap4city/>
- <https://www.linkedin.com/showcase/snap4city-italia/>
- <https://www.facebook.com/snap4city>
- <https://twitter.com/snap4city>

Other channels

- **Linkedin:** <https://www.linkedin.com/in/paolo-nesi-849ba51/>
- **Twitter:** <https://twitter.com/paolonesi>
- **FaceBook:** <https://www.facebook.com/paolo.nesi2>
- <https://www.disit.org>
- <https://www.km4city.org>
- <https://www.snap4.eu>
- <https://www.linkedin.com/company/snap4/>
- <https://www.facebook.com/snap4srl>
- <https://twitter.com/snap4tweet>

I.A. Registration and trial on Snap4City.org

Snap4City.org platform provide a space for trial. Actually, Snap4City.org is multitenant, that means it provided a number of organizations which are separate tenants for data space, applications, dashboards, and users. As trial area we offer DISIT Organization/tenant, which provides a set of representative data and dashboards. If you are registered on an organization/tenant you may only access to data/dashboards published on that organization for the whole public and for the ORG people, and directly delegated to you only. This means that without a specific agreement you are going to see a limited amount of data and dashboards.

Each new user, even in trail, is accepted and registered in the role of **Manager**. He/she can upload data, create proc.logic/IoT App, dashboards, etc., with some limitations. Each user may be leveraged to **AreaManager** role by the administrator to have more power and perform some Data Analytics, etc. Specific functionalities can be only enabled the administrator personally. So that you have o ask, or you organization have to perform an agreement with Snap4City.org, some limitations may be applied as well. According to your capabilities, the system provides you a level of knowledge which describe the functionalities that you have exploited, and thus your awareness of the platform. Your role and level are reported in the login area of the user interface.

These features may not be present in all installations of the Snap4City solution.

I.B. Change of Terminology, for 2023 version of the platform

After many years we are constrained to change some notations: due to the evolution of Snap4City tools and concepts in terms of their capabilities, features, some of the tools and concepts would be better represented with different names. In particular, we are evolving the names as reported in the following table. The change of the name will be progressively propagated in the platform tools, this document, document of technical overview, and on slides, step by step. In the meanwhile, in most cases the text would report both names, and thus please keep in mind that:

Former name	new name, from 2023	What
<u>IoT Device Model</u>	Entity Model	A data model
<u>IoT Device</u>	Entity Instance	A data instance ready to get message for time series
<u>IoT Device Variable</u>	Entity Variable	A variable of an Entity Instance or of an Entity Model
<u>IoT Device Message, device message</u>	Entity Message	A data message
<u>IoT Directory</u>	Entity Directory	The tool for managing models, entities, data models, etc.
<u>IoT Applications,</u> <u>IoT App</u>	Processing Logic	= Node-RED + Snap4City Libraries The tool for visual programming, node-red JavaScript, data flow, ingestion logic, data transformation, data loading, interoperability, business logic.

II. Architectural Flows and Functional Areas

The architectures for IoT (Internet of Things), IoE (internet of Every Things), WoT (web of things) and big data development tools based on microservices can be generalized and can be classified according to the following model in which the most relevant areas and connection among them are identified and described. In **Figure 1**, a *conceptual architecture* is presented in which the most relevant data flows among the main components are highlighted. Data/Control Flows are also drawn by corresponding control flow in the opposite direction, or identical in the case of event driven flows. Each module/block of the figure represents a functional area and not a single tool. For example, the Data Ingestion includes the set of tools, processes and services for data ingestion/production, and the same complexity is present in the Business Logic area, Data Analytics and Front-End areas, and thus also the Storage area may present multiple storages and processing. Identify management and access control are diffused in all the communications among all areas and tools.

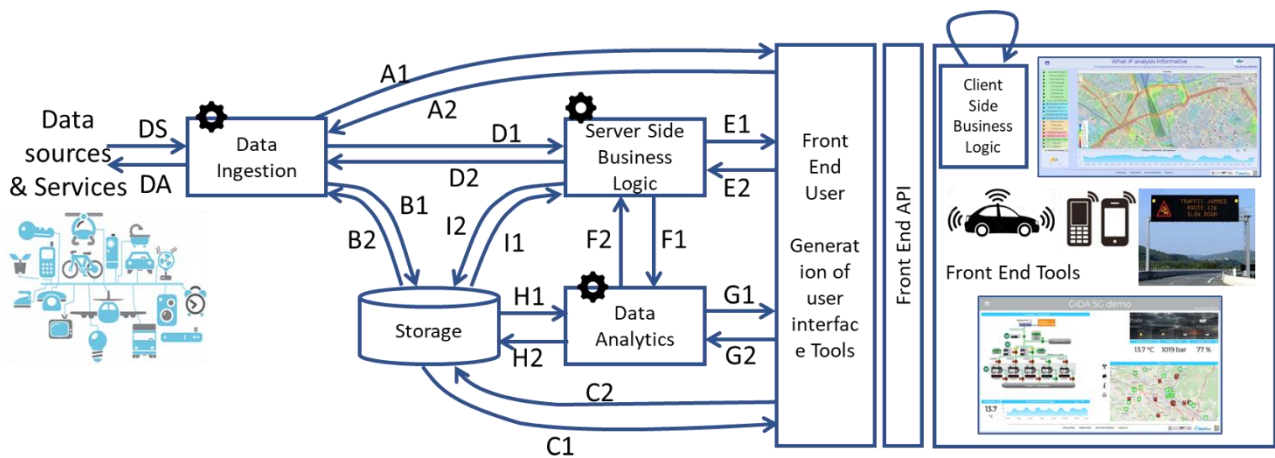


Figure 1 – Conceptual Architecture, from simple to Advanced Smart City Solutions.

The early generation of smart city/IoT applications have been implemented by tools for open data management as well as by GIS (Geographic Information Systems) solutions. In those cases, the IoT data services managed data as Data Sensors / inflows (DS) and Data Actuators / outflows (DA), to store them in some Storage via **B1** (may be indexed and retrieved via B2 or other channel) and rendered them asynchronously via **C1**. Typically, the Data Ingestion / transformation is performed with the so called ETL/ELT (extract transform/load load/transform) processing [MicroServices2019]. The arrival of IoT/IoE devices constrained to be more reactive (data/event driven, managing real-time data) in connecting DS/DA and data services, thus producing the needs of having **A1** connections, possibly data/event driven. In that context, Data Ingestion tools have to be implemented by Brokers (also called IoT Brokers) to manage IoT Devices (Entities) via bidirectional protocols (**C1/2** and **B1/2**). At the same time, for acting on the field, **A2/A1** connection was activated to allow sending data and events from the User Interface to Brokers and Devices/Entities as signals/messages in event-driven mode.

Data Ingestion area is the areas in which a large number of data ingestion processes can be implemented and put in place to cope with many kind of data/entities gathered and produced: Entities Instances/IoT Devices (sensors and actuators, may be based on Smart Data Models, IoT Device Models, etc.), POI (Point of Interest), typical time trends, KPI, heatmaps, constrained scenarios, traffic flows, personal data, user profiles, maps, orthomaps, shapes, GIS data, trajectories, origin destination maps, scenarios, etc. They are produced exploiting a large number of protocols <https://www.snap4city.org/65>.

IoT Edge / Edge solutions are located on the field and before the Data/Entity Ingestion area. They may include a number of the functionalities reported in **Figure 1** (e.g., ingestion, local storage, business logic), while the substance does not change the general data and control flows. Typically, the Edge only includes the

Data Ingestion, which as described above, may include some data/entity transformation and exchange. In the case of Fog computing some Data Analytics mainly far from the Front End and may be not storing the results on some Storage but passing them to some other entity in the network.

Slightly more complex solutions could be those that need to perform some computing on stored data which can be performed only asynchronously due to their complexity (e.g., periodic processes). This problem can be solved by activating Data Analytic processes and services, exploiting connections **H1/2** to read historical data entities and write results back (for example for computing some machine learning, simulation, predictions, early warning, anomaly detection, heatmaps, origin destination matrices, KPI, etc., saving results or derived trained models). The results can be made accessible on the user interface via **C1**, once saved in the Storage Area, or via **I1/E2** if some Business Logic arrangement is needed. In some cases, the results can be passed back to the Business Logic without saving via **F1/F2**. Otherwise, via **G1** the analytics results can be provided on the user interface on the fly without saving values (assuming that the results are just a temporary exploration), that is the classic path of Business Intelligence tools.

Typical smart city applications and services need to compute predictions/simulations, analysis on the basis of historical and contextual data. Examples of smart applications and services exploit computing tools are: *smart irrigation (deciding the best irrigation time), smart parking publishing free parking slot predictions, optimizing routing for waste collection, multimodal routing for final users, and for computing origin destination matrices, heatmaps, anomaly detections as early warning, etc.* The computations could be performed periodically or on demand. The results are usually saved on Storage via communication and services, such **H1/2** connections. The Data Analytics processes may also need to directly interact (via **G1/2**) with the front-end layer in which the user interface content is produced (production of HTML pages, streams, etc.) for the devices such as web and mobile App, Dashboard, digital signage, etc.

Recently, in the context of Smart City, IoT/IoE, industry 4.0, WoT, a number of complex smart applications are demanded by Decision Makers for city management and Control Rooms (with the aim of implementing workflow management, simulation and what-if analysis), developing complex workflows in short time, with frequent updates and modifications of data analytics parameters, data flow, workflow, and business logic / Intelligence on the user interface. To this aim, most of the smart solutions integrated the possibility of implementing a Business Logic module with workflows/dataflows (exploiting **E1/2**, **F1/2** and **I1/2** connections), thus including multiple connections for real time data driven flows (**D1/2**, **E1/2**), as in **Figure 1**. To this end, most of the development environments exploit traditional programming languages (such as AWS, MS Azure IoT, etc.), while in some other cases visual languages and MicroServices are used (e.g., Node-Red by JS foundation [MicroServices2019]). Thus, the capability of the solution highly depends, in practice, on the flexibility of the Business Logic/Workflow, which in turn exploits API and MicroServices, and thus on the integration of the user interface to create smart applications with the development environment and data flow. These smart applications can be regarded as **custom Business Intelligence** tools. In the case of Business Intelligence tool, as well as in Visual Analytics the Business Logic defining the actions in response to the user interaction can be on client side or on server side. **Client-Side Business Logic, CSBL**, allows each user to see its own experience on Front-End without loading the server side of many contextual information for each user. **Server-Side Business Logic, SSBL**, may be realized by taking into account the context of the user interaction or without it. In the former case, it can be used to share the user experience with many users for example in control room and decision support contexts.

Thus, the implementation of smart applications and solutions can be realized by using microservices provided by the functional areas of **Figure 1** and taking into account non-functional requirements. The development framework must enable developers to create advanced applications, and at the same time to support operators to keep the infrastructure under control, despite the complexity generated by multiple applications that may share the same platform, data, data driven processes, data analytic, and users' tools at the same

time.

The **system interoperability** with other solutions can be implemented via a number of API and formats that in Snap4City are more than 150:

- **DA/DS:** In Snap4City, <https://www.snap4city.org/65>
 - API of IoT Brokers,
 - API for data ingestion and data submission,
 - API for GIS,
 - etc.
- **Front-End API** can provide access to Storage, Data Analytics and Business Logic to:
 - **C1/C2** API which are called Smart City API to query data from the storage,
 - **E1/E2** API for Server-Side Business Logic, with WebSocket connections
 - **A1/A2** API with brokers to send data directly on broker, and to recover data from brokers.
 - **Broker, IoT Broker** API, IoT Devices / Entity Instances, IoT Edge Devices / Edge processing tools and devices, etc.
 - **G1/G2** API for exploiting the Data Analytics of any kind,
 - **API and call back for Client-Side Business logic,**
 - **external applications:** web and mobile App, digital signage, navigators,
- **Processing Logic interoperability, which is in Node-RED + Snap4City Libraries of MicroServices:**
 - Any API with **third party services and applications**
 - **API and services of DA/DS** as above described,
 - **Front-End API** as above described.

III. Snap4City Architecture

To fully understand the Snap4City Architecture and tools we suggest to read the document reporting the **TECHNICAL OVERVIEW**: <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf> which also includes a glossary of terms (take in mind the change of terminology as reported in Section 1). On the other hand, a summary is reported in this section for shortening your learning and training.

Snap4City platform is 100% open source, scalable, and modular with a set of tools accessible according to user profiles. **Smart Applications can be easily developed by producing only: data transformation, data analytics and dashboards with almost no coding activities.**

Snap4City can ingest and process/produce data/entities of any domain according to any kind of data sources: GIS (ArcGIS, QGIS, etc.), city utilities (water, gas, mobility, energy, light, water, environment, industry, etc.), legacy systems, personal data, mobile data, database data, IoT Network and Broker, brokers of any kind, KPI/MyKPI, Industry 4.0 protocols and network, social media, telecom data, trajectories, heatmaps, flow, origin destination matrices, satellite data, etc. All sources are bidirectional channels as Snap4City can ingest and produce data/entities with protocols suitable for any channel. See for interoperability <https://www.snap4city.org/65>

Snap4City supports any kind of data flow Entity Networks, communication protocols, and entity/data formats, as well as any legacy and vendor solution. Entity/Data Ingested with any data/entity model are aggregated on the basis of Km4City Ontology into the so-called Knowledge Base (which is the Expert System of the city/industry) and into the Big Data storage, NoSQL. This approach eliminated the problems of data silos and pillars. Any legacy solution, as well as new applications and data/entity is integrated in a unified model, establishing semantic relationships of any kind and respecting the original semantic, thus avoiding the

flatness of data lake that results in poor performance in data/information access when needed for rendering.



Figure 2: Global Architecture of Snap4City solutions and framework

Ingested data, just provided in the Brokers are immediately indexed and stored into the long term storage, becoming searchable and usable for snap4City tools such as the Entity Inspector / Data Inspector, Big Data Analytics, Dashboards builder, Smart City APIs and thus for the Mobile Apps, and via MicroServices for processing Logic / IoT Applications (IoT Apps) which are implemented as Node-RED, used for ingestion, production, integrations, transformation, business logic, server side business logic and intelligence, etc. <https://www.snap4city.org/download/video/course/di/> Data ingestion can be also performed with high performance processes written in Python (provided by Snap4City as configurable script), for example for fast loading historical data from other data base or formats.

Data/Entities can be consumed by data-analytics, Processing Logic / IoT App, mobile App, and Dashboards and Synoptics, third party applications via API and in real time streaming, the so-called data driven processes, and end-to-end secure (from the devices to dashboards, back and forwards directions).

Data Analytics can be developed in Python and RStudio, and the platform provide tools and guide to transform them in containers to be used into Processing Logic flow (IoT App) in Node-RED. <https://www.snap4city.org/download/video/course/da/>

Data Rendering via Synoptics, Dashboards, tables, any kind of graphics widgets can be easily used just selecting the data and choosing the preferred graphical representation of data/entities among the several possible: time trends, maps, bars, multiserries, chords, OriginDestination (OD) maps, tables, heatmaps, trajectories, scenarios, paths, shapes, buttons, dimer, sliders, hierarchies, spidernet, donut, comparing trends, staked diagrams, barseries, custom animated elements, etc. <https://www.snap4city.org/download/video/course/das/>

Developers and qualified operators on Snap4City platform can access the platform tools via web browser

(without any installations on the local computers) to develop solutions and applications in the Snap4City collaborative environment exploiting Data Analytics, Dashboards, and Processing Logic / IoT Apps (for data ingestion, adaptation, transformation; business logic, data analytic management, etc.). Data Analytics developers can also develop their application on their local computer if they prefer.

Mobile App developers need a local development Environment depending on the target mobile devices. Final users can access to Snap4City services via Web and mobile devices, dashboards, synoptics, interfaces for digital signages, tables, panels, etc.

A Snap4City Living Lab allows stakeholders to collaborate for the production of smart solutions and to the innovation of the whole smart ecosystem. Snap4City provides a methodology for stimulating the innovation identifying the most relevant and effective changes and solutions according to a quadruple helix approach. A Living Lab with the web based Snap4City Development Environment provides a comprehensive set of tools for developers and stakeholders to implement data ingestion and processing flows, Data Analytics algorithms, Dashboards, Processing Logic / IoT App, Synoptics, Custom Widgets, and Web and Mobile Apps [BlgDataService2018].

Smart Applications can be easily developed exploiting the cloud infrastructure by producing only: Processing Logic / IoT App, Data Analytics and Dashboards with almost no coding activities. Orange parts of Figure 3 are those usually developed, while all the rest is part of the provided microservices and infrastructure. Third party applications can dialog with the platform and each other, and can be integrated with the solutions via Smart API, or via Brokers or via Processing Logic / IoT App any protocols.

Third party applications can dialog with the solutions via:

- Smart City API, Swagger: <https://www.km4city.org/swagger/external/>
- Broker/IoT Brokers API, for example for NGSI context Broker, Entity/IoT Directory: <https://www.km4city.org/swagger/external/?urls.primaryName=Orion%20Broker%20K1-K2%20Authentication%20API>
- Processing Logic/IoT App any protocols: <https://www.snap4city.org/65>
 - And they can also expose some specific API, custom made.
- Authentication and Authorization APIs.

Edge and Fog processing can be implemented by using Processing Logic / IoT App or other means (custom processes), and may have direct connection with Snap4City Microservices on cloud or locally installed features as local devices, local database, local dashboard, local html pages, etc. API can be those mentioned Smart City API, Broker API. For example, an Edge processing tools may be implemented with Raspberry Pi, windows, Arm, Linux, etc., in which Node-RED is native or can be installed and Snap4City library can be loaded/installed as well.

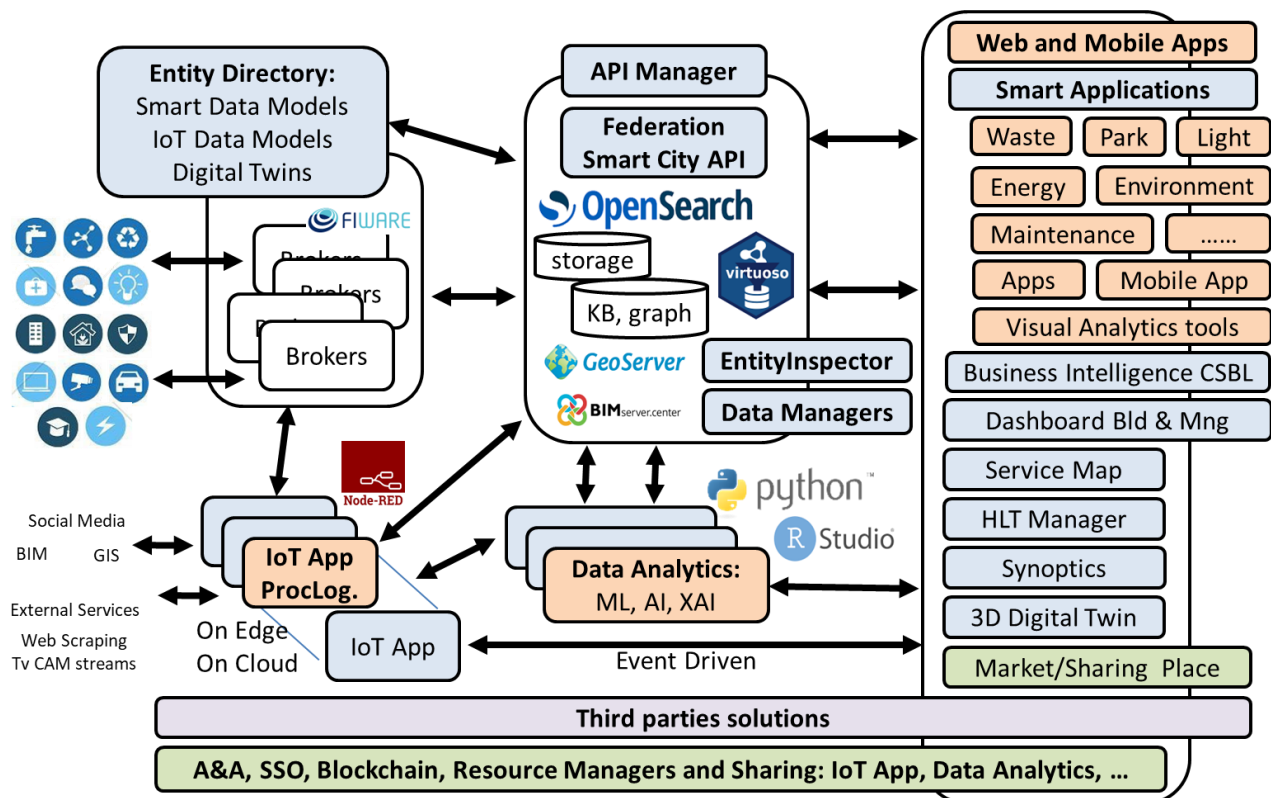


Figure 3: Architectural Overview of Snap4City framework

In particular, for the *Smart Applications' Development Life Cycle, SMADELC*, presented in this document we refer to **Figure 3** from the left side:

- data ingestion/transformation and interoperability.** Data may enter/exit in push via brokers of different kinds, and/or via dedicated processes in pull/push (enabling the data/event driven approaches). Data/Entities may be listed in some Entity Directory (IoT Models, Entity Model, Entity Instances, Smart Data Model of FIWARE, data spaces).
 - Data/Entity may enter/exit from in push via Brokers of different kinds.** All the brokers are managed by the Entity Directory / IoT Directory which registers them and registers any kind of Data/Entity Model including the Smart Data Model of FIWARE. The brokers can be internally managed (internal Brokers) or external which are managed by third parties. The integration of external brokers with their data/entity model and devices is automated, fast and easy. Please remind that brokers can be also used to act on entities/devices on the field.
 - Data arriving in the platform are stored automatically on the Storage** (Open Search) and / or can be sent directly to the user interface or to data analytics for stream processing. The same flow works as data driven from entering data/entities to dashboards and from dashboards to data streams for acting and sending commands, alarms, etc. Data collected and stored can be browsed by using the **Entity Inspector / Data Inspector** and **Data Managers (HLT)** for all the High-Level Types managed in the platform. From Entities, IoT, POI, maps, heatmaps, traffic flow, etc.
- Processing Logics** could be in some ETL/ELT script. In Snap4City, **Processing Logic / IoT App** are Node-RED plus Snap4City microservices or Python processes (to push data in NiFi which is the gate to the storage, OpenSearch). Visual programming languages such as Node-RED, NiFi are typically preferred for rapid prototyping. These processes can be also allocated on cloud, fog, edge and on premise/field.
 - Data/Entities may enter/exit in push or pull via processing Logic / IoT App** as described above to manage data flow from any kind of protocol and format. Please remind that brokers can be also used to act on devices.

- **Data arriving in the platform are stored automatically on the Storage** (Open Search) and / or can be sent directly to the user interface or to data analytics for stream processing. The same flow works as data driven from entering data/entities to dashboards and from dashboards to data streams for acting and sending commands, alarms, etc. Data collected and stored can be browsed by using the **Entity Inspector / Data Inspector** and **Data Managers (HLT)** for all the High-Level Types managed in the platform. From Entities, IoT, POI, maps, heatmaps, traffic flow, etc.
- **storage and semantic models and reasoners.** In Snap4City, the arrival of a new entity/data model does not imply for the developer to adequate the storage structure, but to model it on the Entity/IoT Directory once. Once a new Entity Model is registered, any instance can be directly deployed, and specific data loaded. Entities/Data entering in the platform can be automatically (i) stored (in **Figure 3**: Open Search), (ii) semantically indexed, (iii) directly sent to the user interface, or to data analytics for stream processing. The same data flows work as data driven coming from dashboards to data streams for acting and sending commands, alarms, etc., in the platform and/or to external devices via brokers or processes. Data stored (in some cases, the storage is called data shadow) can be browsed by using the Entity Inspector/Manager, for all the entity/data types managed in the platform. From simple data entities to IoT devices, maps, heatmaps, flow, matrices, 3D interactive representations, scenarios, etc., as needed by the DT.
- **Data Analytics** can be developed in Python, RStudio, MapReduce (only in the presence of Hadoop based platforms), etc., and can be activated from: (i) Processing Logic / IoT App (Node-RED), (ii) scheduled processes, (iii) data driven events; (directly on cloud or on developers' computers). Data Analytics may exploit stored data via some API to access at the data in the storage and may implement any kind of solution based on machine learning, AI, XAI models, tool, library or dedicated hardware you need, as NVIDIA, etc.
- **data representation front-end generation with server-side and client-side business logics.** User interface of smart applications can be easily developed by creating HTML generation tools, such as Dashboard Builder [Dashboards2019], which allows to select the graphics views and connect them to data/entities on storage and to data streams and brokers. Graphic widgets can be configured and scripted to create business intelligence tools by using JavaScript on client- and/or on server-sides. In the latter case, the business logic can be implemented via Processing Logic as Visual programming Node-RED. A number of tools are available for the customization of the user interface: dashboard builder editing and composition, wizards, synoptics development tools and templates, templates for the user interface themes, micro applications, etc.
 - **Smart Applications** can be easily developed by creating the user interface with the Dashboard Builder, which allows to select the graphics views and connect them to the data of the storage and to the data streams. It is possible to glue the graphic widgets with JavaScript on client side or demanding the implementation of the business logic on server side or a mix. Server-Side Business Logic is implemented by using Processing Logic / IoT App via visual programming. A number of tools are available for the customization of the user interface: dashboard builder editing and composition, wizards, synoptics development tool and templated, templates for the user interface, ready to user micro applications, etc.
- **All the data streams (internal and external) are protected, authenticated, and authorized according to [Security2020], Pen Test passed, GDPR compliant, etc.**

A **Conceptual version of Snap4City architecture is reported in the next figure**, the main elements of which are reported and described below.

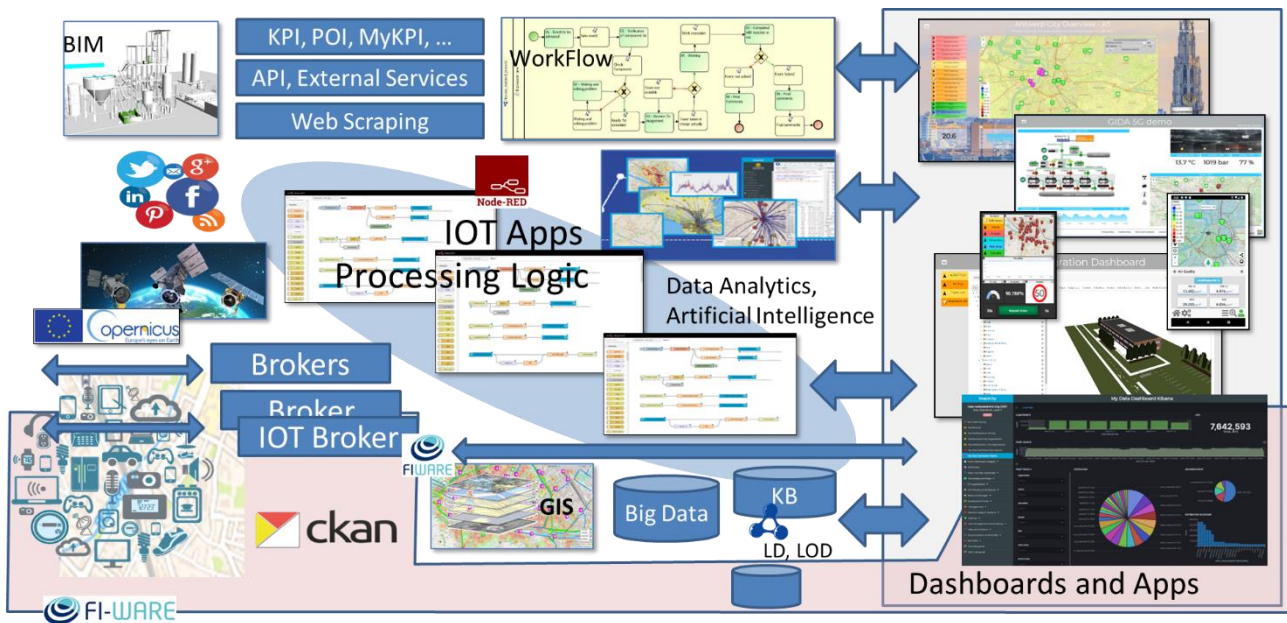


Figure 4: General Concept from <https://www.snap4city.org/download/video/course/gen/>

Conceptually, from left to right of the above figure:

Data Ingestion, transformation, aggregation and unification, as well as **Data Publication** are performed in push/pull by using: <https://www.snap4city.org/download/video/course/iot/>

- **Processing Logic / IoT Apps** in Node-RED and Snap4City Libraries/palette to perform data ingestion, data transformation, integration, data storage, business logic, and implementing ETL and ELT (extract transform load, extract load transform), including the exploitation of a large number of protocols: WS, FTP, REST Call, etc.
- **Brokers**: FIWARE IoT Connectors, IoT Agents, IoT Adapters, Brokers, IoT Brokers, also integrated with Processing Logic / IoT Apps with dedicated MicroServices and the **Entity Directory** / IoT Directory,
- **Data Table loader** and POI Loader for Excel files ingestions including IoT Device Data (Entity Instances) and POI, for bulk data ingestion, also integrated/developed with **Processing Logic** / IoT Apps which can be customized for each organization or for multiple table/excel kinds,
- **Tools** developed in Python/RStudio or other languages which can exploit the Smart City API of Snap4City for data ingestion, transformation, and registering Entity Instances from Entity Models, etc. Some of these tools are accessible as open-source processes to be customized,
- Satellite data service using dedicated Python processes also integrated with **Processing Logic** / IoT Apps,
- data gathering tools such as **DataGate/CKAN** for open data (<https://ckan.org>) are also integrated with IoT Apps by using dedicated MicroServices,
- GIS data ingestion/publication via WFS/WMS, also integrated with **Processing Logic** / IoT Apps; for example, using GeoServer, ArcGIS, etc.,
- **Web Scraping via Portia**, also integrated with **Processing Logic** / IoT Apps.

In our experience, **99% of the data ingestion/transformation processes can be easily implemented in Processing Logic / IoT App Node-RED and managed with one or more Processing Logics / IoT Apps**. Processes can be internally scheduled, and automated backup of node-red flows and versioning can be performed using Node-RED features. Only high throughput processes should be activated by using Python that can be controlled by **Processing Logic** / IoT App and deployed in Container, also managed by Snap4City infrastructure, or can be actually put in execution in other manners. The Python processes proposed by

Snap4City can expose REST API. A template for developing Custom Tools in Python is also provided by Snap4City. For Performance analysis please see the installation page: <https://www.snap4city.org/738>

In the context of **big data architectures**, the storage is a facility which does not need to be internally designed in terms of tables and relationships among tables (as in the old-style software engineering approach). All the data entities are modelled in terms of: Smart Data Models, Entity Models, IoT Device Models, etc. Therefore, in Snap4City, all the data models are **Entity Models** / IoT Device Models, also the FIWARE Smart Data Models are present into Snap4City as **Entity Models** / IoT Device Models which is a higher level of model abstraction and brings automatically the relationships into a knowledge base which is a semantic graph database (an evolution of reticular databases). So that any data record in the big data platform has a definition in terms of **Entity Models** / IoT Device Model, which is a **Data Model**. Therefore, in the context of IoT, IoE and WoT Snap4City manages different entities such as: FIWARE Smart Data Models, Snap4City **Entity Models** / IoT Device Models, **Entity Instances** / IoT Devices, custom devices, and IoT Brokers/Brokers, that are registered via **Entity Directory** / IoT Directory, a multiprotocol multi-broker tool for IoT Network management. Among the brokers, a major role is played by Orion Broker of FIWARE by which the platform support NGSI V1, V2, and LD with Services/Tenant and Service Paths. The **Entity Directory** / IoT Directory is capable of automatically deploying Orion Brokers on demand. You can connect/register any Broker to Snap4City **Entity Directory** / IoT Directory. In this case the broker is regarded as an External Broker, not controlled by the Snap4City platform. External Orion Brokers can be harvested for registering their data on the platform in a fast manner, thus reducing connection times to existing infrastructures. A detailed description of data capabilities and networks is provided in: <https://www.snap4city.org/download/video/course/iot/> . The Snap4City platform supports a very large number of protocols push/pull, sync/async. Among them are MQTT, NGSI, COAP, OneM2M, ModBus, OPC, WMS, WFS, and AMQP. To get a larger list of supported protocols see <https://www.snap4city.org/65> .

For **Data Formats**, the Snap4City supports a large range of **High-Level Types**, *HLT: Entity Instances, IoT Devices, FIWARE Smart Data Models, Entity Models, Sensors/Actuators, POI, Trajectories, Origin Destination Matrices, Flows, Traffic Flow, People Flows, Heatmaps, Satellite data, 3D BIM building, 3D shapes of city, social media, routing, public transport offers, geographical shapes and grids, KPI, TV Camera streams, predictions, animations, synoptics, city scenarios, events of several kinds, user profiles, custom formats, etc.* A Digital Twin is an HLT instance and may present several connections to other data kinds, which in turn are instances of other HLTs. Each Digital Twin can be accessed via the so-called Data Inspector.

To remark, the Entity Models (IoT Device Models) can be used as template to create Entity Instances (IoT Devices). The Entity Instances (IoT Devices) are those that can receive messages to update their data over time, and thus to create time series. The Entity Instances and thus most of the instances of the High-Level Types need to be referred from other entities or to see them on Dashboards, etc.

References to Entity Instances are established via the so-called **ServiceURI** in the Knowledge Base, Service Map (also called Device URI in Entity Directory / IoT Directory). We can define the **ServiceURI** as a URI identifier of an Entity in the solution and it is defined according to the international standard definition of an URI: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier In Snap4City, **ServiceURI**, also called **SURI**, for example as: <http://www.disit.org/km4city/resource/iot/orionUNIFI/DISIT/METRO759> if you put the SURI on a browser and the SURI refers to a public entity, its definition appears since it is compliant with the Linked Data standard.

A large number of details regarding data ingestion are reported in <https://www.snap4city.org/download/video/course/di/> while details regarding data formats are reported in <https://www.snap4city.org/download/video/course/das/> which provides evidence that for each data formats several graphical widgets can be used for its visualization together with other kinds of data.

The Big Data Storage is managed by multiple solutions:

- **Time series** from Entity Instances / IoT Devices and MyKPI are automatically feed into the storage Open Search cluster (which is the new name of Open Distro for Elastic Search of AWS) for storing and indexing data.
 - The Entity Instances / IoT Devices need to be registered into Entity Directory / IoT Directory, which in turn automatically registers them on Knowledge Base for reasoning.
 - The MyKPI can be also stored into some SQL database.
- **Knowledge Base**, KB, (based on Km4City ontology) implemented as an RDF store (Virtuoso) which is an index for geospatial, relational, and temporal aspects. The KB can be federated, for example for federating different KB or different organizations. Whenever a new data model (Entity Instance or Model) is registered in the system, the registration is performed into the KB. Different instances of the KB can be federated via Smart City API by creating a mutual connection among cities/areas of the network, if the single installation wants to share, and deciding what to share.
- **Heatmaps, Orthomaps, Traffic Flows Maps, and Maps** are managed by the Heatmap Server which is a GIS (for example GeoServer or ArcGIS if you have one installed with WMS facilities) and can distribute the maps according to WMS/WFS protocols towards Web and Mobiles App and Dashboards. They can be loaded from Processing Logic / IoT App using specific API.
- **Origin Destination Matrices** are managed by the OD/ODM Manager and represented into the Multi Data Map widget in dashboards. They can be loaded from Processing Logic / IoT App using specific API.
- **Internal Buildings 3D Shapes, and Floors** are managed and distributed by BIM Server. Standard BIM tools are used for editing and interchange in IFC formats with standard tools such as AutoDesk Revit, etc. Buildings and Floors are shown in dashboards for their integration with maps and time trends of IoT devices. The BIM are managed on Snap4City by the BIM Manager and can be loaded from the user interface using IFC format.
- **3D City Representation** can be loaded according to different components from which they are composed such as: image patterns, building shapes, 3D shapes, LIDAR data, traffic flows, POI, IoT Devices, etc. [DigitalTwin1], [DigitalTwin2], [GeneratingDigitalTwin]. They need to be loaded in agreement with the snap4city platform manager into the Dashboard area.
- **Etc.**

Processing Logic / IoT Apps can be used for a range of activities in data ingestion, storage, interoperability, transformation, data driven, etc. The activities of **Processing Logic are better and widely described in other sections**. They are also used for implementing user interface Server-Side Business Logic behind Dashboards to implement smart applications, business intelligence, including interactive Widgets, Custom Widgets and Synoptics, which can be very useful in Smart City Control Rooms. Therefore, with the **Processing Logic / IoT App** it is possible to create end-to-end applications which can get event driven data from the field and rendering data on a dashboard, as well as receive some command from the user to act on **Processing Logic / IoT App** processes and maybe act as actuator. These aspects will be recalled later on this document in the context of Dashboards. <https://www.snap4city.org/download/video/course/das/>.

Data Analytics are processes written in RStudio or Python that can: (i) perform data access (data taken from outside or from the Storage/KB of Snap4City Platform), (ii) apply statistics, operating research, Machine Learning, Artificial Intelligence, Explainable AI, deep learning tools, reinforced learning, algorithms also exploiting Tensor Flow, CUDA, Keras, Pandas, etc., (ii) produce results as predictions, early warning, hints, new data, heatmaps, anomaly detection, etc. Snap4City Data Analytics in RStudio, Python and IoT App can be developed by using offline and/or online development environments which also allows putting in execution of the processes in Containers which expose their API/MicroService for Processing Logic / IoT Apps and other tools in the platform. Off-line development environments are viable as well as, if preferred. The API could be made accessible from external applications if mapped on Firewall. With Data Analytics processes

you can produce/compute: predictions, early warning, traffic flow reconstruction, classifications, clustering, trajectories, anomaly detection, KPIs and indicators, Typical Time Trends, typical trajectories, routing and paths, multimodal routing, travel plans, query results, simulations, analysis, calibrated heatmaps, smart parking suggestions, car sharing analysis, what-if analysis, automated ingestion of satellite data, social media processing and sentiment analysis, etc. A large number of success stories are provided in: <https://www.snap4city.org/download/video/course/da/>

Smart City APIs (SCAPI) are the main entry point for accessing data from Big Data Storage (KB and Open Search). They include a large collection of services to: exploit queries and reasoning on the storage and Knowledge Base, access/control IoT Network, exploit Data Analytic results, etc. All the data and services are accessible via the Smart City APIs which are used by front-end tools such as Dashboards, Web and Mobile Apps, and MicroApplications. Details regarding Smart City APIs are reported in: <https://www.snap4city.org/download/video/course/app/>. **Smart City APIs provide a large set of facilities** to get data filtering them by: area, path, GPS locations, distance; date and time interval; value of the variable of the models; Service URI of the entities, etc. They API are authenticated and provide data according to the authorization accesses of the user as described in the following. So that, they are very useful for implementing services from Mobile App, etc. The Snap4City platforms can be federated each other via Smart City APIs. The Smart City API can be also exposed via some API manager. For the purpose, Snap4City suggests APIMan open-source tool.

Advanced Smart City APIs, (ASCAPI) may include Smart City APIs and all other APIs for: Entity Directory / IoT Directory management, Orion Brokers direct access (if needed to get the last data and perform subscription on the eventual NGSI events), heatmap management, flow management, dashboard management, user management, process management, etc. Advanced Smart City APIs are used to federate Snap4City platforms, are documented with Swagger, and can be controlled for accounting data access and billing according to different business models. The Advanced Smart City API can be also exposed via some API manager. For the purpose, Snap4City suggests APIMan open-source tool.

Dashboards can be created by using the Dashboard Builder and/or Open Search Dashboard (former Kibana, sister of Grafana) for different kinds of users such as: decision makers, city operators, ICT operators, private users, etc., suitable for Smart City Control Rooms with video wall, for city Operators on Desktop of multiple monitors, mobile operators, and Situation Rooms with touch panels. Dashboards can exploit all kinds of data, HLT, and Data Analytics, legacy services, and special tools such as traffic flow reconstruction, decision support systems, what-if analysis, scenarios definition, etc. Dashboards are created by using a large range of ready to use Widgets (for rendering data time series, data on a map, interacting with data and tools, providing visual analytics), and also Custom Widgets creating Synoptics as SVG elements and animations, or exploiting other graphic libraries, such as D3.js, Highcharts JavaScript, etc. <https://www.snap4city.org/download/video/course/das/>

III.A. Considerations and Remarks

We hope that, after the above introduction of Snap4City solutions and development environment, it is clear that the developers in Snap4City can exploit its capabilities of a rapid prototyping and with reduced needs of coding for developing applications and smart applications:

- **You do not need to Design the database with the design of entities and relationships** since the data models are called Entity Models / IoT Device Model from which by Entity / device registration the instances are created. The data associated with Entity Instances (IoT Devices) may vary over time creating complex versioning and time series of each entity keeping trace of any kind of system and data evolution. Devices via their messages, time series can refer each other and create any complex pattern you need, more sophisticated relationships that you could create in classic relational databases.
- **You do not need to implement from scratch visual data representations** since the connection from Big Data storage and visual rendering tools is already provided as optimized solution, just to be used from some Business Logic, or just connecting data to the graphic widget ready to be used. Thus, the user interface designer can focus on the structure of the dashboards, user interface and client-side business logic if any.
- **You do not need to design complex queries.** In the world of big data, SQL is quite impossible to be used exploiting complex queries in which the explosion of temporary structure would create impossible processes to be implemented and would not be scalable. So that, the data models and structures with their complex relationships have to be well prepared to be exploited and scalable, all the relationships defined at level of model, devices, and instances can be efficiently managed by the provided semantic database. Thus, the optimized queries and filtering are provided ready to use via
 - Processing Logic / IoT App as MicroServices
 - Smart City API REST calls to be used from Dashboard automatically via Dashboard generator.
 - Smart City API REST calls to be used programmatically from Web and Mobile Apps

Therefore!

Smart Applications can be developed by producing: Processing Logics / IoT Apps, Data Analytics and Dashboards with almost no coding activities on the functional aspects and for server side.

Coding may be needed into:

- Processing Logic / IoT App Node-RED in JavaScript only
- Dashboard widget Client-Side Business Logic if needed in JavaScript only
- Data Analytics: if needed and realized in Python and/or RStudio

Smart Applications Developed on Snap4City implies to create your artefacts and smart solutions, decision support systems, business intelligence tools, etc., in fully control of your Intellectually Property and not open source.

You Applications and IPR in Snap4City are composed by:

- **Data Models:** Entity Models / IoT Device Models, Smart Data Models, etc.
- **Proc.Logic / IoT App:** data ingestion, adapter, transformation, wrappers, business logic, transcoding, integration, interoperability, algorithms, etc.
- **Data Analytics:** algorithm and processing in RStudio or Python, ML, AI, XAI, etc.
- **User Interface Design:** Dashboards, client-side business logic, Synoptics, widgets, templates, styles, etc.
- **Client-Side Business Logics (if any)** realized in JavaScript on Dashboard widgets.
- **Server-Side Business Logics (if any)** realized in Processing Logic as Node-RED and JavaScript.
- and the **data instances** for the High-Level Types.

The fact that you have developed your solution in Snap4City (which is 100% open source) does not imply that the artefacts and your smart solutions are open source as well. They are 100% under your control, and you can decide how to release them, which kind of licence to impose on them. This also implies that Snap4City has no rights on what you have developed using Snap4City development tools. In fact, it is totally equivalent of developing a program with any other open-source development tool. For example, if you develop a database with MariaDB, the tables and queries are not open source; another example: if you develop something on NiFi Apache which is open source, your flow is not open source; another example: if you develop something on ECLIPSE which is open source, your Java/PHP/C++ Code is not open source, etc.

If you are developing some smart solutions on some platform labelled as *Powered by Snap4City* technology, the platform manager and owner may have specified some rights on what you have developed according to the terms of use THEY expose on their own platform, please read them in any case. For example, the rights to off and discharge your solution if it creates some problems or violate the rules imposed by the service. For example, ethical rules.

IV. Development Life Cycle

The **SMADE-Ic** approach can be classified as an AGILE method as reported in **Figure 5**. The **development process is performed via a number of incremental Sprints, in which at each cycle/sprint the completeness of the solution is increased**. The first sprints are usually dedicated to the implementation of stand-alone applications in a distributed complex system with simple data rendering just for monitoring the data ingestion and providing the evidence of the functionalities, that usually belong to the final applications as well. Then, in successive sprints, modules/processes are combined by adding data analytics and more complex data rendering and business logics. Final *sprints* are dedicated to the integration and general optimization to pass in production via final validation.

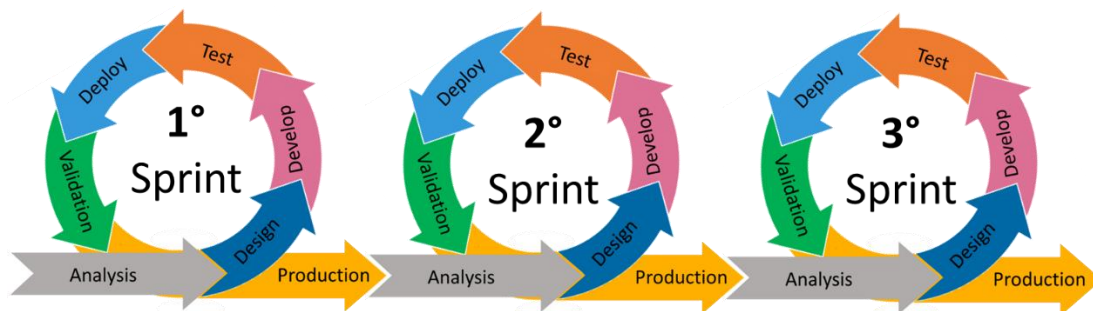


Figure 5: Agile like development life cycle for IoT smart solutions.

The main activities phases of each sprint are organized as in **Figure 6** and are described in the following. Please note that, since the aim of the approach is fast prototyping the phases of Analysis, Design, Development, Testing/Deploy, Validation and Production may be performed at the same time on different modules/components of the whole system. In each sprint, the activities of Analysis (Plan), Design, Develop, Test (Review, regression testing), Deploy (Launch) may be performed by different specialized teams with some interchanges of information regarding ready to use data, data analytic, and user interfaces.

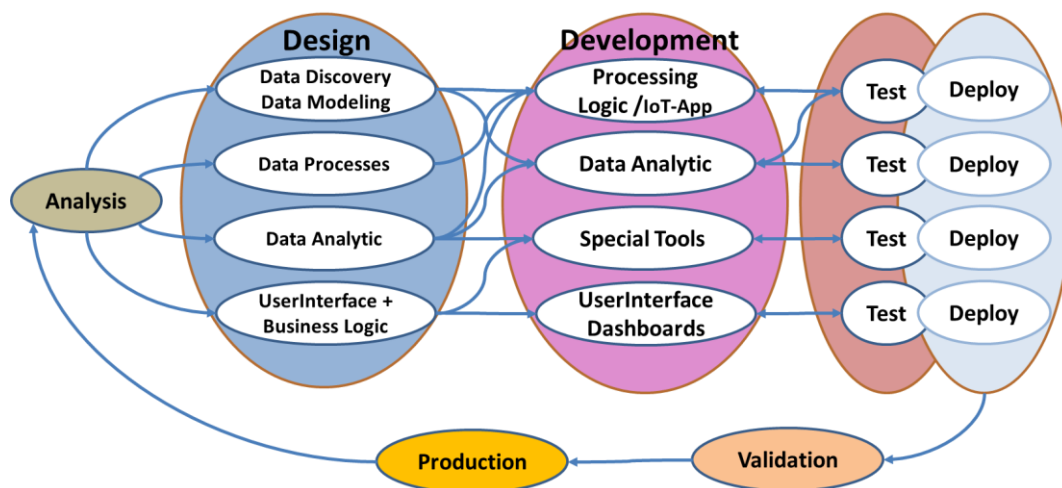


Figure 6 – Single development cycle/sprint from analysis to production.

For example, when some needed data sources have been identified, the corresponding development of data / entity ingestion can start, and, at the same time, the analysis on the other parts could start as well. Another example: the data scientists can start developing data analytics based on the accessible data, while the user interface experts could start the design and develop of the user interface parts as dashboards on the accessible historical data, even if not all data has been ingested, etc., as soon as the data kind have been

understood. Therefore, the activities of Test&Deploy for each unit can be performed at the same time and may lead to rapidly reach the planned goals of the current sprint, providing the results to the other teams and passing to the next goal and sprint.

According to Snap4City platform, **Figure 7** reports the mapping from **Snap4City Tools** on the development life cycle phases and activities. The names reported are **Snap4City** tools and/or main items in the Snap4City menu on Snap4City.org platform and they may be present in *Powered by Snap4Tech* platforms according to the platform size/model decided and features installed. In **Figures 7**: the “**Tables and Documents**” module refers to the indications provided in this document regarding Tables and Documents to be produced in the phases of Analysis and Design.

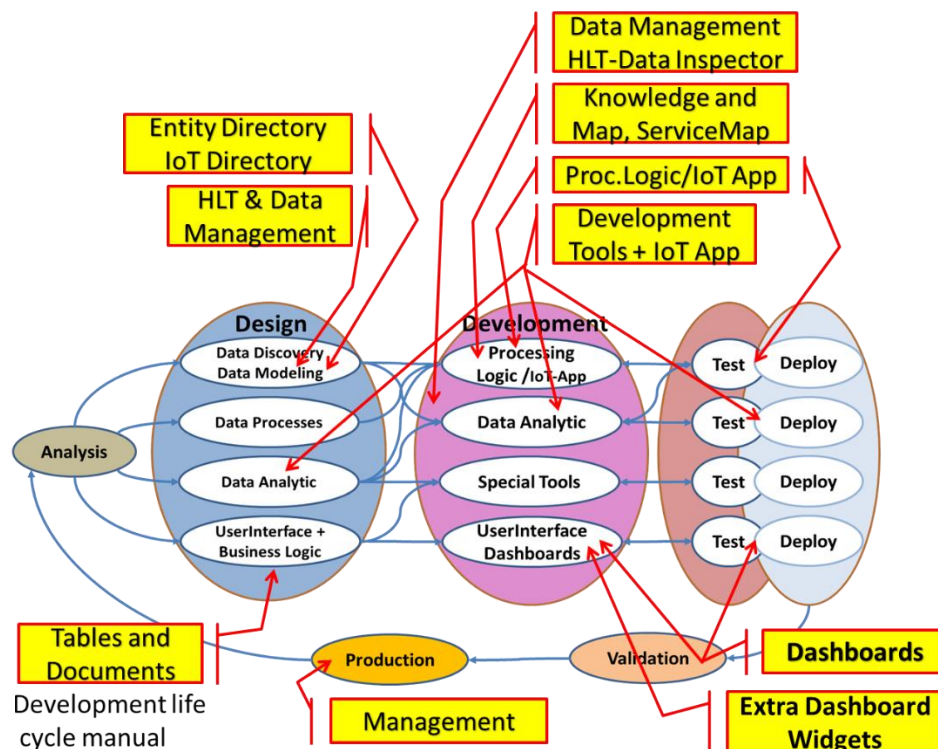


Figure 7: life cycle with annotated Snap4City tools.

Special Tools can be developed in any programming language, and can exploit the services of Snap4City platform, tools and microservices via the above-mentioned API. In addition, any web-based user interface of Special Tool can be embedded into the Dashboards via the External Content Widget as IFRAME. The development of a smart application may imply the need to develop some Special Tools. In addition, any web-based user interface of Special Tool can be embedded into the Dashboards.

The Validation & Production phases (also for each sprint) lead from partial to full integration of modules of the progressively implemented smart applications. Please note that, in the context of Snap4City rapid prototyping, the: (i) processes for data ingestion, transformation, are Processing Logic / IoT App (which are node-RED) which can developed in collaborative manner, cloned, shared and executed in containers on cloud, and can be also deployed on Edge, (ii) the Data Analytic processes are coded in Python/RStudio which can be cloned/shared, multiply deployed and executed in containers, (iii) user interfaces of smart applications are typically based on dashboards which can be cloned/shared, and also embedded in third party web pages. This approach allows to plug any components dynamically and to pass from development of modules (with local Test&Deploy passed) to their validation in the integrated context and thus to arrive on production in smooth and progressive manner.

IV.A. Analysis Phases

Analysis: phase in which the comprehension of the problem/subproblem, extraction of requirements and definition of the main elements for the specification is performed.

Please take in mind that the analysis can be performed by subsystems or just incremental for adding new features to your former solution (previous Sprint or actually legacy solution). Thus, you do not be afraid to produce since the first-round a full set of exhaustive requirements, or scenarios, or use cases. We suggest you, at the first sprint to focus on the most relevant aspects and start design them and develop the prototype with those elements only in the first sprint.

The main activities of the Analysis Phase are:

- workshops and discussions on the track of Innovation Matrix by domain, and Entity identification
- production of the Dictionary via entity identification: physical, virtual and modules/tools
- Scenarios formalization
- Use Cases formalization
- Requirements formalization
- Sequence Diagrams formalization, for the most relevant aspects

- This phase should answer at questions such as:
 - What the solution/task should do?
 - What the users/entities are expecting to get from the solution/task?
 - Who is going to use the solution/task?
 - How many users/entities are going to use it at the same time?
 - Which is the most preferred modality to interact?
 - Which are the most effective representations of results?
 - When is it going to be used?
 - Which are the KPI to confirm that the development has reached the goals?
 - ...

This phase is successfully performed by:

- **Performing workshops** for Innovation Matrix by domain: mobility, energy, parking, gov services, etc. (see **Figure 8**).
- **Entity Identification:** which is the **Dictionary** to be used into the Use Cases and Scenarios. Classic entities and specialized entities are respectively as follows. The entities are the major data and actors in the description coming from the workshop for the Innovation Matrix. Example of entities could include:
 - **Actors and their profiles (as Data Models):** User, Operator, final user, ICT expert, decision maker, doctors, driver, etc.
 - **Entity Models:**
 - **Entities and their digital counterpart (Data Models)** for: Vehicle, Analysis, Server, Client, Mobile App, parking area, etc.
 - **Actors** are modeled as **Entity Model** if they need to be represented into the system.
 - **Entity Instances:**
 - **IoT Devices which are instances of the models** as: City user XX, Control Room Operator, Doctor Rossi, Cop 3726, Car FI796HG, IoT Device XY, Trip 34, Patient Health Record for Robert, etc.
 - **Modules or Tools** of Third party or legacy tools: they are applications, servers, IoT Edge subsystems, well known services for data providing, gateway, brokers, etc., which should interact somehow with your solutions. They can be on cloud or on some premise, they can provide you some External API, of some kind: WebServices, Rest Call, FTP, Web Socket, MQTT, etc.
 - **Tools:** which can be actual software or hardware tools, and also data analytics, algorithms, procedures.

- **External Services/Tools/API** which can be provide by third parties. They can be HW/SW tools to interoperate with Modules and among each other. For example, external brokers, gateways, third party APIs, etc.
 - **External API:** to interoperate with any other application and service.
 - **External Services:** to host into the user interface and Dashboards elements coming from third party applications.
- **Scenarios Formalization** describing the application/task, textual definition, with some standard table as UML. The scenarios have to refer to identified entities.
- **Use Cases Formalization** describing the different cases into the single applications, by using UML formalization, there are specific Use Cases for each Scenario. Please focus on the most relevant, those that are adding value to your solutions. The others can be given for granted in a first phase.
- **Requirements Formalization and elicitation** via workshops and documents analysis by using standard tables, prioritizing them, setting mandatory/preferred/optional, functional and non-functional, first/second/third release, etc.
- **Sequence/Activity Diagrams Formalization:** to highlight some of the critical aspects. For example, to describe the user interaction, and/or the interaction among main entities, putting in evidence which is the Entity starting the dialogue with respect to the others involved (e.g., a client requesting data to the server, a device sending data to the broker). UML sequence diagrams are a suitable formalization for the purpose.

IV.A.1. Innovation Matrix by domain, and Entity identification

Snap4City Innovation Matrix

Parameters		Commons
Current State	Needs				
	Current Practices				
	Value proposition (current)				
Future State	Value proposition (Future)				
	Solution				
	Value Capture				
	Key Partners				
	Barriers				

Figure 8. Example of Innovation Matrix.

Starting from the development of a different **Innovation Matrix for each domain**: mobility, energy, parking, gov services, etc. (see **Figure 8**). The goal is to identify the current practices, their problems, the possible improvements foresee, eventual criticisms, and also the assess the risks connected to the application of changes to present modalities of doing the activities, and thus the effective pros and cons of any possible or requested change. The approach is a specialization of the Innovatrix of [<https://timreview.ca/article/1225>]. Details on the compilation of the Innovation Matrix can be recovered from <https://www.snap4city.org/download/video/course/sys/> and its detailed versions presented at the Snap4City workshops.

As a first step the where for each column (of the last tree on right in table of **Figure 8**) the main **Actors** are identified. For example: operator, city users, decision makers, etc. The compilation can be performed during a workshop in live or on some open board online, such as MIRO platform. The workshop should involve: Experts of the domain specific, Experts of different customers segment, Operative people, ICT people, Decision Makers, etc. The **Innovation Matrix** table is compiled by using stickers during workshops in which the moderator is pushing the group to answers to the following questions.

- **Current State**

- Needs
 - How do we prioritize these needs?
 - How many times per day they need those services?
- Current Practices
 - What are the current practices to satisfy the above needs?
 - What are the pains/cons of the current practices?
 - What are the gains/pros of the current practices?
 - Is there any signage about the need to change current practice?
- Value proposition (current)
 - How much (measurable) is used the current solution?
 - Why is it used or so diffused?

- **Future State**

- Value proposition (Future)
 - What could be accepted, what it should be available with the new solution with respect to the current, ... the benefits?
 - What (measurable) impact would you need for the customer segment?
 - Which could be a measurable index of success, with reference value?
- Solution
 - What are the components of a possible solution, app, Dash, etc.?
 - How do these components differ for the different customers segments (which kind of profiling)?
- Value Capture
 - What value (non-monetary/monetary) do you receive in return from customer segment?
 - Short vs long term value capture?
- Key Partners
 - Who are your key partners (unit, team, subcategory)?
 - How to interact with stakeholders (how do you plan to promote, make evident, make it available and accepted)?

- **Barriers**

- What are the barriers for adoption, usage?
- Is a workaround possible?
- How will we work around these barriers?

As a result, **for each domain**, that means for each **Innovation Matrix** table, a document section has to be created according to the following schema (for each Actor or for the first consumptive column):

- Target Segments:
 - users and their characterization.
- Benefits for identified user categories.
- Actions towards them and the possible solutions
 - Variations on Actions.
- Other Actions to be verified: non clear aspects if any, verification needed to finalize the actions to be performed.
- Data Identified:
 - the data that could and would be the first to be ingested,
 - understood and verified if they are suitable for the purpose.
 - Any missing data?
- Means of Communication with user categories for promoting the new approach.

IV.A.2. The Dictionary via entity identification: physical, virtual and modules/tools

The Analysis via the Innovation Matrix allows to perform the identification of Entities (or better of the Entity Models), and from Entity Models one should be capable to create a Dictionary to develop scenarios, use cases, requirements and some sequence diagram. We suggest “Some” since to develop all of them may not be needed since the beginning, we are in the agile model. Please identify only those that are the most relevant as: data ingestion, system integration, user registration, data analytics, etc.

For example: *Let us now to suppose that we have to develop a solution for monitoring Vehicles and Drivers. Each Vehicle has a profile description and can be driven by a number of drivers over time. **Each Vehicle** can experience some maintenance and performs trips in the city area. A trip has an official start/end and over time is described by its velocity, acceleration, brakes, charging level, or tank level, etc. **Each Driver** has a profile and can use a number of Vehicles to perform trips. During the trip also the Driver is monitored for its healthiness, attention, etc., and before, during and after the driving, periodically or sporadically may experience some Analysis to certify its capability to drive in that moment and for the next days. The Driver may experience some warning cases for healthiness, some tickets from policeman, some warning for high-speed velocity or generically bad driving, some problems from the vehicle's status, etc.*

We identified Entities Models such as: **Vehicle, Driver, DriverHealthiness, VehicleEvent, DriverEvent, DriverAnalysis.**

The phase of Analysis has identified entities/models classified into

- *Physical Entities / Devices* (**token, on board unit on the vehicle, device for assessing the drive attention, etc.**). All *Physical Entities* are sending data on the platform on server via some IoT Broker or other data ingestion process, for example via some local application, third party service, IoT Edge solution, etc.
- *Virtual Entities / Devices* (**Vehicle, Driver, DriverHealthiness, VehicleEvent, DriverEvent, DriverAnalysis**). *Virtual Physical Entities* are managed on server side for managing data and providing a set of user interface views for data entry (registering data) and for data rendering visualization.

See **Figure 9** for example. All of them, in Snap4City, can be managed as Entity Models, and implemented as Entity Models / IoT Device Models for their definition. Once defined they need to be actually created/instantiated via registration. Once registered they (the device/entity instance: user45driveranalysis)

can be used to collect data into the storage by sending messages to the Device instance (for example on: user45driveranalysis).

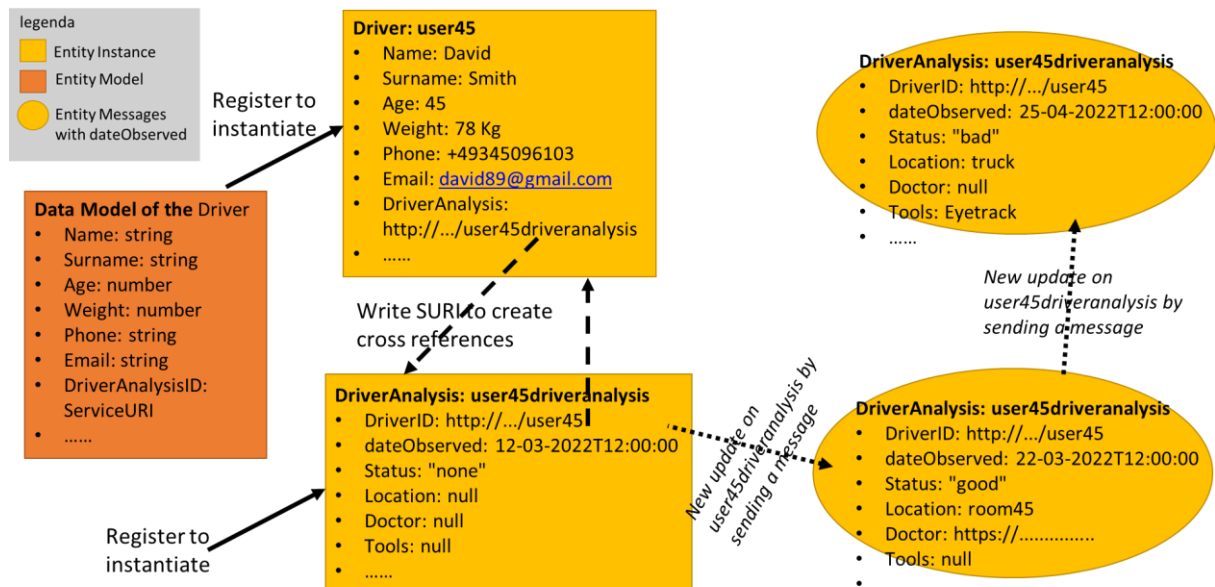


Figure 9: Example of relationships among Entity Model (IoT Device Model), Entity Instance (IoT Device), Entity Messages (IoT Device Messages), according to the Snap4City terminology of 2023

Another example: The solution has to provide a User's interface for registering: Vehicles, Drivers, and Driver Healthiness, and when produced also result of computed Analysis for driver healthiness on some period of time. In the moment in which a Driver is registered (e.g., user45), one should also register the personal DriverAnalysis Entity Instance (IoT device) to be prepared to collect user45 analysis (Entity Messages) performed in many situations over time. For example, as in **Figure 9**. Monitoring data for real time assessment are performed by Entity Instances (IoT Devices), special Data Analytic performs the Driver Healthiness assessment and provides the assessed score to the Driver Healthiness of the specific Driver. The Analyses may change the status of the Driver confirming that it is enabled to drive or should be pushed to take a rest. Driver Events of **start** of a Vehicle usage, and of **end** of Vehicle usage are registered by the Driver which passes its token on the Vehicle. Critical events (e.g., DriverEvent such as a ticket to the Driver, **VehicleEvent** a Ticket for maintenance of the vehicle, the detection of critical conditions, etc.), are registered as events sent to the systems, from some Entity Instance (IoT Device) or via some operator user interface or computed by some Data Analytics. Monitoring and interacting on the view regarding Vehicle and Driver evolutions and status on the basis of integrated views taking into account Physical and Virtual Entities have to be possible. Please note that, according to the sequence of messages (see **Figure 9**), it can be observed that the user45 has changed its status over time. This is what is going to be reported on dashboards and also used in the analysis.

The analysis of the above description allowed us also to identify a number of Entity Models which are neither Physical or Virtual Entities, which would be realized as **major entity modules/tools in the solution** (which in turn are realized as: **Processing Logic / IoT App on cloud or on Edge, Dashboards, Data Analytics processes**, and eventually **Mobile Apps and Web App** using Smart City API). For example:

- Mobile App for the Drivers.
- Operator tool for registering actual Drivers (as Entity Instances of the Entity Model: Driver) and monitoring their status
- A number of Data Analytics processes:
- A number of edge processes as Processing Logic / IoT App in Node-RED or other means
- A number of data collection Processing Logic / IoT Apps, ...

- Etc.

We suggest, to provide them a name as soon as possible and to enter all Entities in a **Dictionary** organized as a table. The dictionary should collect both the Data Models and the Modules / Tools you are going to develop in your solution.

Dictionary					
Term	Entity Model or Module	Kind	Responsible	Status	Spec where
Driver Healthiness	DriverHealthiness	Entity Model	Dr. Rick Ross	To be done	To be defined
User profile A	DriverA	Entity Instance			
Vehicle Event	VehicleEvent	Entity Model			
Remote Console	MyOperation	Application	J.T. Kirk	To be done	lost

The columns in green are related to the design phase, which may be not fully specified/filled in the analysis phase.

The legenda for the columns is:

- **Terms** for example in Capital Case for the first letter, or CamelCase as you prefer (we suggest not use spaces and neither special characters since most of the programming languages and brokers do not like those characters). For example: Vehicle, Driver, DriverHealthiness, VehicleEvent, DriverEvent, DriverAnalysis
- **Entity Models or Module:**
 - **Data Model:** FIWARE Data Model, IoT Device Model, custom model, smart data models, etc.
 - **Modules could be specific, for example:**
 - Processing Logic / IoT App: **Ingestion data from devices etc.:**
 - for different purposes as described in the rest of the document
 - Dashboard: **operator user interface**
 - Client-Side Business Logic on widget dashboard
 - Server-Side Business Logic on Processing Logic in connection to Dashboard
 - Data Analytics: **estimation of drive condition algorithm SK43**
 - Third party tools:
 - Third Party API:
- **Kind:** Entity Model, Entity Instance, Processing Logic / IoT App on cloud or on Edge, Dashboards, Data Analytics processes, Mobile Apps and Web App, third party tools, etc.;
- **Responsible:** who has to develop and/or maintain
- **Status:** inherited, already accessible, to be customized, to be finalized, to be done, in cloud, on edge, etc.
- **Spec Where:** location of the description, of the specification. It can be a document or a link.

External API: have to be identified and listed to allow to interoperate with the platform and among partners/developers.

Each external API should be:

- Listed with:
 - entry point, kind of protocol, parameters, examples
 - authentication model, credentials if any
- **SWAGGER** definition: <https://swagger.io/>
- **Postman:** <https://www.postman.com/>

Please note that Postman need to have JSON data on the same line.

External API						
API name	API url and shape	Kind	parameter	Credentials approach	status	Description, Swagger link, Postman, ...

The columns in green are related to the design phase, which may be not fully specified in the analysis phase.

In Snap4City, External API of third-party services can be:

- Easily called via Processing Logic / IoT App which are Node-RED, in formalizing processes of data ingestion, data transcoding, transformation, business logic, etc.
- Wrapped into a MicroService which can be exploited into Processing Logic / IoT App.

In Snap4City, dedicated custom API could be created from Processing Logic / IoT App.

This practice is strongly discouraged for a number of reasons reported as follows:

1. Custom made APIs from Processing Logic / IoT APP should be avoided,
 - a. They are not secured. they are not authenticated and neither support the authorization control for their access. API provided from Processing Logic / IoT App, would not to be easily secured, and very difficult controlled into the general SSO of the system.
 - b. The interpretation of the REST call parameters should be performed into the Processing Logic / IoT App and also the search of entities and filtering.
 - c. They are not scalable in terms of multitasking support.
2. Most of the API that you are going to create in Processing Logic are typically created to provide data results may be requesting them to the storage. So that, the same data can be easily obtained by using Snap4City Smart City API, which are in place, authenticated, controlled in access and scalable.

Moreover:

Any API call that you need to access at Data Time Series, data collections, and data instance status (on the basis of some variable and filters) can be easily implemented by using Snap4City Smart City API.

- See for smart city API call and filters in **Section IV.C.4.**
- See for authentication before calling API in snap4city in **Section IV.C.5**

Please use Smart City APIs which provide:

- **a large set of facilities** to get data filtering them by area, path, GPS locations, distance; date and time interval; value of the variable of the models; IDs of the entities, etc. They are authenticated and provide data according to the authorization accesses of the user. So that, they are very useful for implementing services from Mobile App, etc. Thus, increasing flexibility and simplifying the solution.
- Authentication and authorization mechanism, see **Section IV.C.5**
- Robustness to the cybersecurity attacks, reliability
- documented in swagger, see **Section IV.C.4**
- Support for scalability

Please note that mobile app and third-party tools accessing to Snap4City API need to have a ClientID (which is going to identify the kind of client is approaching the platform, for safety reason) registered on KeyCloak, please ask, see Section IV.C.5.

External Services can be used to insert into the user interface and Dashboards elements coming from third party applications. **They have to be listed as:**

- URL, parameters, size of the screen

- IFRAME constraints

Please note that in Snap4City they can be easily listed and, in a click, integrated into Dashboards.

External Services				
URL	parameter	Description	Nature	Subnature

IV.A.3. Analysis: Scenarios formalization

The scenarios are formalized by using tables. A UML Activity diagram graphically represents a process (or Scenario). <https://www.uml-diagrams.org/activity-diagrams-examples.html>

According to the above example, the **Main Scenarios** to be described may include:

- The operator registers the driver into the system (the driver is requested to register on the platform before the actual registration on the monitoring activities, thus including the signed consent for GDPR).
- The Driver takes the Vehicles for day trip: starts, drives (while being monitored), ends.
- The Driver is assessed in the analysis center, and results are saved and processed allowing or blocking him to drive.
- The Driver may be monitored on its activity of driving to provide direct real time feedback.
- The operator analyses the last 3 months of a Driver.
- The operator analyses the usage of Vehicle.
- Etc.

The Server/platform has to communicate some notification to the Mobile/Web App in real time. For some cases it could be done with some Web Socket as provided from Snap4City platform and Dashboards, otherwise in most of the cases the communication with third party applications and/or special purpose mobile apps should not need to be in real time. Thus, messages from Operator/client mobile to Platform could be performed by sending messages modelled with some Entity Model but on a specific Entity Instance by sending an Entity Message to a platform broker. Notifications can be internally performed with some changes into the Entity Instance status.

Please note that Mobile App cannot be easily pushed, they have a dynamic changing IP and they do not have web server of APIs to be called easily. The Mobile Applications cannot have a process listener active in real time since the IP address on mobile is going to change, some of the operating system does not allow to have a process active on the mobile phone to reduce the power consumption and the user security. Both Apple and Samsung provide some service to send message in push on the basis of the mobile phone number/id. See for example, APNS (Apple Push Notification Service) of Apple, Airship, Firebase messaging, wonder push, etc.

A simple solution is to develop the mobile app to make periodic polling on the server (dashed lines) to get notifications. If this approach is not viable, there are some services of Apple, Samsung, or third party which use the services of the mobile operator. Another possibility is to send a message to the Driver via SMS, or Telegram. SMS is passing from the mobile operator.

Therefore, since most of the clients are on mobile App. This means that if the data analytics or decision support is taking time and thus the communication back has to be performed Asynchronously from the

mobile to the server → that is **Asynchronous interaction** among elements, or using some services, or a developing platform using an external service.

So that, the Server/platform can change the status of some IoT Device (for example, for remarking that a notification is ready, variable: "NotificationXXYYReady"), and the Mobile App can periodically call the IoT Device via Smart City API (authenticated and authorized) to understand and get the instructions to be performed according to the NotificationXXYYReady, for example a questionnaire. Once the action on mobile app has been done, the Mobile App can patch/post on IoT Device via NGSI V2 patch/post API a new message including the resulting data of the action.

If you have hundreds or thousands of users and actions we suggest staying on **Asynchronous interaction**, which is more scalable.

Example: Activity Diagram

- **Continuous Lines** can denote event driven, sync communications... for example by sending data on IoT Broker
- **Dashed lines** can denote Pull data collected periodically. Mainly Async. Communication from Platform to Mobile Devices
- Coloured Dots are the different devices data storage

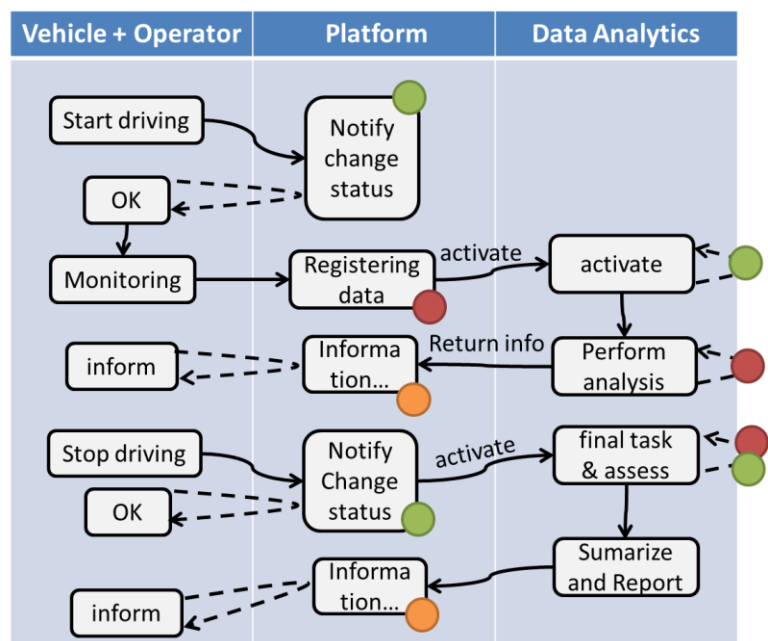


Figure 10: Example of Activity Diagram

The typical interaction from some Web or Mobile App can be based on sending messages to the platform via IoT Orion Broker in NGSI V2 and collecting results asynchronously requesting them via Smart City APIs. In both cases the access to the API (of IoT broker and of the Smart City API), has to be performed by using open id connect. This means that the Web or Mobile App has to get the token and then connect. The Smart City API can also be acted without authentication to get public data.

In **Figure 10**, the mobile app or the car is notifying to the platform that is starting or beginning is driving turn with a message to the IoT Orion Broker (the message on the Orion Broker will be automatically feed into the storage). If the message is not sent to the Orion Broker but on MQTT, the message needs to be decoded (in most cases) and remapped to an NGSI V2 format. This can be performed in an IoT App on platform.

The Mobile App can immediately ask to the platform server to get the status by using the Smart City API (to verify if the start state has been registered) and may be to get destination or other travel trip information, among those, also the ALERT if the driver cannot drive for example.

Once the mobile app received the authorization to start, the mobile is starting monitoring (position, moving velocity, assessment of the driver, etc.). This is information is provided in push on IoT broker. The Platform will register all of them. A process on the platform, running on IoT App and/or on Python, can analyze the data collected and decide to continue support the driver in driving or informing him to stop or to be more

careful. The Mobile App should be in foreground to continuously get information by polling (dashed line) from the direct reading of some device status using Smart City API.

A variant of the above **Figure 10** can be **Figure 10b** as follows.

Every time a data is entered into the Storage an event occurs into the broker. The server «Inform» can be subscribed from an IoT App to receive in push these changes (red dashed line). To this end, the Snap4City Library of microservices provide a node for subscription on broker.

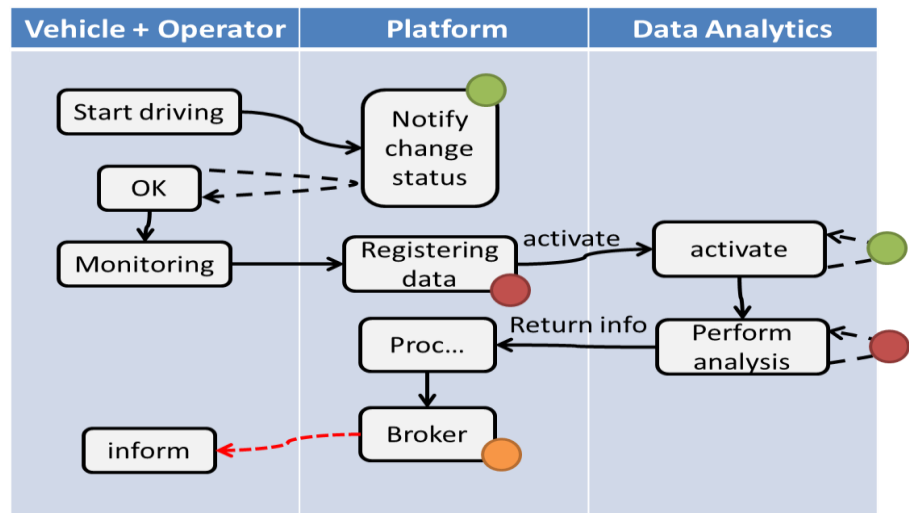


Figure 10b: Variant of Activity Diagram of Figure 10.

In **Figure 10b**, an example of Sequence/Activity Diagram is reported. The driver on its Mobile App, he/she marks the start of the driving section, and the App notifies the change of status to the platform via some broker, once performed all the needed verifications (taking some minutes, may be). The effective change and authorization to start is made accessible by the platform to the mobile app which is requesting the status in pull (dashed line). Then the mobile app starts to monitor the drive status continuously, and send new data (e.g., the level of attention, the road taken, etc.) to the platform via some broker every minute. The arrival of new data may activate some data analytics to perform some analysis of the collected data (red dots) and producing results on the platform data. In the case in which the process detected critical conditions for the driver, the assessment procedure on platform may decide to send an event/message (dashed red, in push from platform to clients) to the operator and driver via a Broker to warning the driving of the lack of attention or for some wrong path. The event in push from platform to client could be a viable approach on some platforms and may have some limitation on Mobile App in which the interaction paradigm can be changed in a periodic REST call from the Mobile to the Platform.

For instance, as explained in the training slides, according to **Figure 10b**:

- the **black continuous line** (push) will be used to send some data on the platform broker with a REST call which has to be Authenticated and Authorized according to the OpenID Connect as explained in Section IV.C.5, and would be in the form of:
 - for Orion Broker NGSI V1, which may accept POST:
 - <https://<platformdomain>:8443/orionbrokerfilter/v1/updateContext>
 - For Orion Broker NGSI V2, which accepts PATCH on Snap4City protected Orion Brokers:
 - <https://iot-app.snap4city.org/orion-broker-xxxxxx/v2/>
 - Or in the form for non TSL protected interaction NGSI V1 (for example, each platform and organization may have different end-point for Smart City API):
 - <http://iot-app.snap4city.org:80/orion-broker/v1/updateContext?elementid=ELEMENTID&k1=K1&k2=K2>
- Please note that**
 - the use of K1, K2 is discouraged in favor of more secure solutions **see Section IV.C.8**),

- the **black dashed line** (pull) will be used to request some data from the platform by using a REST call to smart city API (Authenticated and Authorized according to the OpenID Connect as explained in **Section IV.C.8**), in the forms:
 - via regular Smart city API by category, etc.
 - http://svealand.snap4city.org/ServiceMap/api/v1/?selection=59.581458578537955;16.71183586120606;59.62875017053684;16.875171661376957&categories=Street_light&maxResults=100&format=json
 - Via Super
 - <https://www.disit.org/superservicemap/api/v1/?.....>
 - This case is valid for **SUPER on snap4city.org**.
 - Via Super by values
 - <https://www.snap4city.org/superservicemap/api/v1/iot-search/?selection=43.77;11.2&maxDists=700.2&model=CarPark>
 - <https://www.snap4city.org/superservicemap/api/v1/iot-search/?selection=42.014990;10.217347;43.7768;11.2515&model=metrotrafficsensor&valueFilters=vehicleFlow>0.5;vehicleFlow<300>
 - This case is valid for **SUPER on snap4city.org**.
 - Please note that:**
 - each platform and organization may have different end-points for the above APIs.
- the **red dashed line** (push) will be used to send some data from the platform (from an Orion broker) to some stable IP client or other machine for machine-to-machine communications:
 - As a first step the client has to subscribe to some entity on the Orion Broker passing its IP where the broker will have to send the data in push
 - This example refers to NGSI V1 which is deprecated and substituted by V2**
 - NGSI V2 examples are reported in Section IV.C.6.**
 - The POST will be in the form of [/v1/subscribeContext](#) passing as parameters: elementid (the device ID, and K1, K2 (please note that the approach with K1/K2 is mainly deprecated)) or TSL approach
 - [curl -X POST](#)
<https://broker1.snap4city.org:8080/v1/subscribeContext?elementid=myspersonaldataster-device&k1=4e0924a8-fdd6-49cf-8d4a-f49cb5710d8b&k2=240567da-64a4-43b3-8ac9-1265178f3cbe> -H "accept: application/json" -H "Content-Type: application/json" -d '{"entities":[{"type":"Ambiental","isPattern":false,"id":"myspersonaldataster-device"}],"attributes":["temperature"],"reference":"http://prova/","duration":"P1M","notifyConditions":[{"type":"ONCHANGE","condValues":{"temperature"},"throttling":"PT10S"}]}'
 - Please note that**
 - the use of K1, K2 is discouraged in favor of more secure solutions **see Section IV.C.8**,
 - each platform and organization may have different endpoints for this API
 - Then the broker will send the messages to the subscribed client
 - it could be possible to have this kind of push also by using Kafka and/or WebSocket, but this is possible with simple and direct exposed API to all Snap4City platforms.

The external APIs of Snap4City are documented in Swagger

<https://www.km4city.org/swagger/external/index.html>

In alternative, for creating some **Synchronous user interaction**, it could be possible to:

- Generate HTML pages from Processing Logic / IoT App and allows to access to some form from that HTML page (Server-Side Business Logic)
- Generate HTML pages from Processing Logic / IoT App and publish them including forms on some External Service Widget on Dashboards (Server-Side Business Logic)
- Generate a form using Processing Logic / IoT App widgets elements FORM, and expose this form on a Dashboard (Server-Side Business Logic)
- Generate Dashboard with an External Service widget formalizing the form in its internal JavaScript Addition (only for professional users) to implement Client-Side Business Logic.

IV.A.4. Analysis: Use Cases formalization

Use Cases can be represented by tables and by sequence diagram

(https://en.wikipedia.org/wiki/Sequence_diagram).

In the above example, possible use cases **may be**:

- Save (register vehicle) /modify Vehicle profile and status.
- Save (Register Driver)/modify Driver profile and status.
- Save an Occurrence of a VehicleEvent.
- Send the data regarding a certain DriverAnalysis.
- Assign and Taking a Vehicle to start driving.
- Close a period of driving.
- Save DriverHealthiness.
- Save Occurrence of a DriverEvent.
- Observe the trajectories performed by a given vehicle and related velocity, acceleration brakes, etc.
 - Identify which driver has drawn a specific vehicle.
- Observe the time trend of a given Driver in terms of healthiness, attention level, etc.
- See the list of all Maintenance Tickets of a given vehicle.
- See the list of last 6-months DriverAnalysis of a certain Driver.
- See the time trend of the last 6-months DriverAnalysis of a certain Driver.
- Etc.

IV.A.5. Analysis: Requirements formalization

The **Functional Requirements** could be listed by starting from the identified active Entities (which could be human, Tools, autonomous processes (for example the assessment process)) and describing for each the single functional statements. Collect all of them by using the following table on which technical details can be formalized when ready:

- Users, Drivers, operator
- Vehicles
- Events of different kinds
- ControlDashboard, ControlPanel
- RegistrationInterface
- Measuring device
- Etc.

Requirements						
ID	Main Entity / Area	Description	Relevance / Priority	Main Tool / Entity involved	Status	Source Code
D1	Operator	The Operator has to be authorized to register Drivers	mandatory	OperatorTool	Not developed	JavaScript

D2	Driver	The Drive can verify its registration by putting Password to access to its data on the solution	optional	Web and/or Mobile App accessible for the Drivers	accessible in open source	Yes In Java

The columns in green are related to the design phase, which may be not fully specified in the analysis phase, where:

- ID: a unique identifier
- Main Entity / Area: the most relevant entity involved or the main area, a mixt of them should not be used.
- Description:
- Relevance / Priority: mandatory (a must), optional (a should), other values could be coded. For example: high, mid, low, very low, etc.
- Etc.

In addition, a number of **non-functional requirements** may be also specified detailing the solution behavior with respect to aspects of: security, privacy, scalability, reliability, availability, etc. Please note that Snap4City is compliant with a large number of Non-Functional Requirements. For these reasons, please consult snap4city portal to see the level from which your solutions developed with Snap4City would natively start. Snap4City is GDPR compliant, respect privacy in the structure, is secure since passed PENTest and vulnerability test, is modular, scalable, interoperable, compliant with a large number of standards, open being open source, portal since you can install on multiple platforms, open being highly interoperable, user friendly having a nice and easy to use user interface, user friendly since you can customize the user interface, etc.

Using Snap4City you can avoid to massively take care about compliance with NON-Functional requirements in your analysis and design and thus you can concentrate on your solutions. This does not mean that you cannot create some critical cases, for example publishing all your user profiles' data violating the GDPR or implementing local API in IoT App without any authentication and control. Snap4City provides a high flexibility, to avoid critical implementation please read guidelines suggested in this document and in the training slides: <https://www.snap4city.org/577>. Snap4City also provide clear performances bounds for each installation of the platform and for the typical MicroX installation. See them on <https://www.snap4city.org/738>

IV.A.6. Analysis: Sequence Diagrams formalization

Sequence Diagrams: describe the interactions https://en.wikipedia.org/wiki/Sequence_diagram behind a certain use case.

For the same recurrent example:

- Driver, driver attention device, Broker/platform, Data Analytics, **DriverHealthiness**.
- Etc.

For example, in **Figure 11** the sequence diagram describing the update of the Vehicle profile and status is shown. As can be seen, we supposed to have two types of possible update, the first manually performed by the user, the second performed after receiving an asynchronous message from an IoT Device through an IoT Broker. This use case includes entities for User, Vehicle, IoT Broker, and IoT Device installed on the vehicle. The IoT App (see more later) represents the business logic that interact with the Vehicle stored representation.

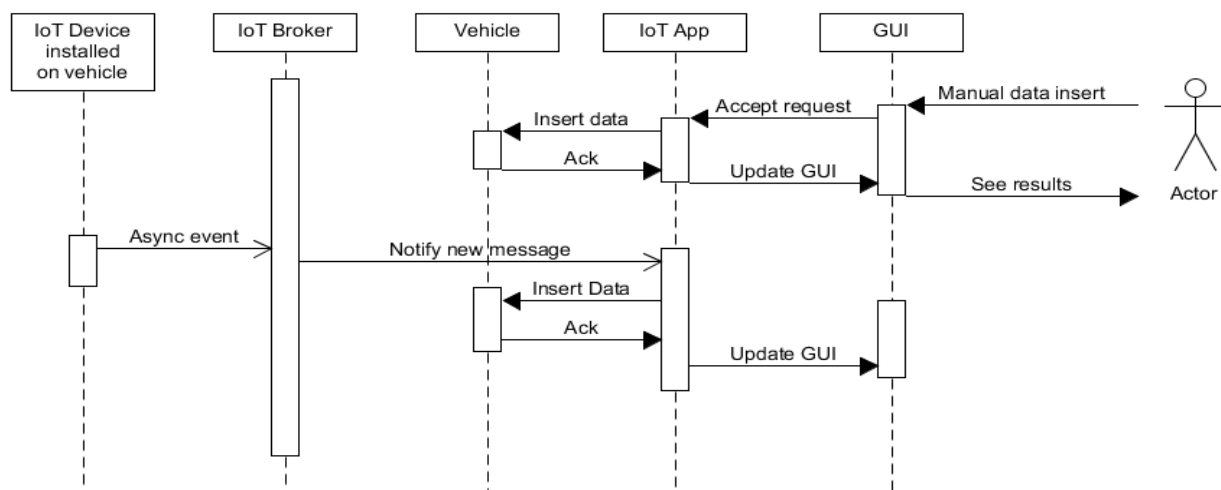


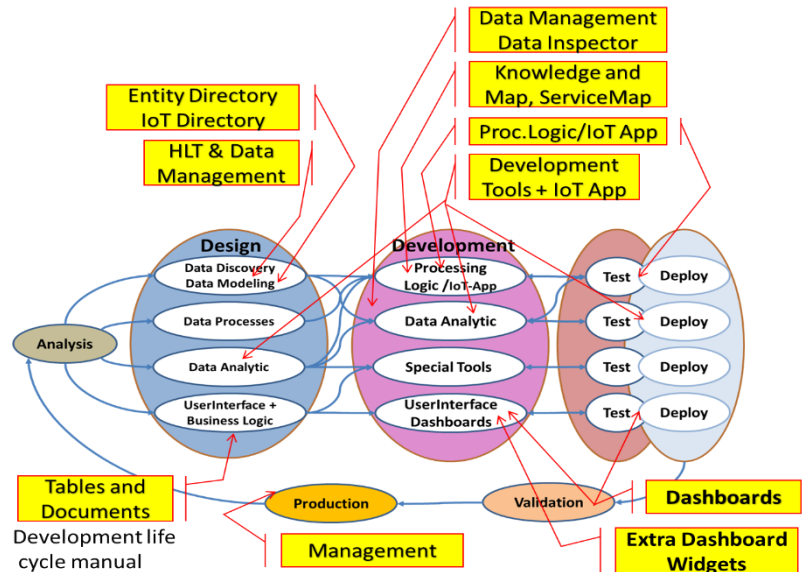
Figure 11: Example of sequence diagram, IoT broker is any Broker, IoT App is a Processing Logic

As a general remark: Do not worry if at the first sprint of the above steps you forgotten to fill some details. It is quite sure that, you have also provided some details that would have to be revised/changed at the next iteration. The suggestion is start developing from the core parts, which are the production of Entity Instances from the Entity Models, the ingestion of Entity Messages for the Entity Instances, etc., and detailing the most relevant and innovative Use Cases with respect to the state of the art. They would leverage the smart solutions to a new level.

IV.B. Design Phases

Design of the Several Aspects:

- Design: **Data Discovery**
- Design: **Data Modelling** → formalization of Entity Models
- Design of **Data Processes** including data ingestion, data transformation, production, publication, etc.
- Design of **Data Analytics**
- Design: **User Interface Design** and Business Logic: dashboard design, and user interaction design



In the schema on the right side, the Snap4City tools which can be used during the design phase are highlighted. In the following SubSections a guideline for each Design aspects is provided.

IV.B.1. Design: Data Discovery

An important aspect to be considered during the design phase is the Data Discovery. This activity deals with the need to identify data, entities, HLT, (historical or real-time) to be collected from the field, from the context, from external/internal, national or local databases, from satellite data, from Open Data (CKAN), or from other networks or Gov portals.

The data to be discovered are:

- **Not those** generated internally by the devices/entities of the solutions.
- **those** needed for the implementation of the solution and that should be accessible from outside the solution. They can be historical data or real time data connectors. For example, the contextual maps, the distribution of vehicles, the user profiles to be involved in the experiments, etc. Among them also the open data can be point of interest, POI, maps, road graph from open street map or gov databases, cost of gasoline, cost of energy, etc.

This phase has to answer at questions such as:

- Where can I take the data, I need?
- There is any kind of data that can be a good surrogate of the data required?
- How they can be accessed?
- Which kind of license they have? Are those data private/public, which licensing?
- Is the license associated with them functional to the solution purpose or not?
- Which information is present in these data, it fits the purpose?
- Is the licensing compatible with the purpose?
- Are those data ethically compliant?
- Do I need to create a DPIA for GDPR?
- Do I need to establish and sign an agreement with some data provider?
- etc.

How to proceed:

1. Data Identification is performed on the basis of the Entities identified and of the needs of Data Analytics / Transformations. The discovered data may have their own data model, and thus the model is going to be adopted in the corresponding design phase.
2. The developers should verify if the needed data are available on the (Snap4City) platform, or they are accessible elsewhere and integrated in some manner and how, etc.
3. If the needed data are missing the phase of **Data Ingestion Development** has to be addressed otherwise one could directly pass at the IOT App or to Dashboard Development.
4. Before the Data Ingestion the data agreements have to be developed and signed (on the basis of the data licensing), verification of Data Ethics, and development of GDPR compliant procedure as the DPIA. According to the data agreement, the enforcement of rules can be performed in the Snap4City platform and business logic, if needed.

Please note that in most cases, the needed data could be surrogated with other kind of data. For example, sensors data for the environment monitoring can be surrogated with satellite data with some connections; CO2 data with traffic flow data, NO2 with traffic flow data, map data with OSM, Orthomaps there are multiple providers, open data are accessible and most of them describing the contextual situation can be recovered from different providers in local, regional, national, and international institutions, etc. etc. As the last possibility one could even buy some the data from Mobile operator, Google, TomTom, Here, Insurances, municipalities, public transportation operator, bike sharing operator, etc. etc.

The process of data discovery in most cases collects data which are associated with data sets and not only to the single Entity. The data for data sets are related to the owner, acquisition model and protocol, format, volume, rate, etc., and for the purpose a table or excel file is used, even if this information can be saved in the Digital Twin, data inspector of the platform.

Description	domain	S/CO/RT	I/O	Type	Status	Referen t	Provid er	endpoi nt	Authent ication	HL protocol	protoc ol	HLT	Format	Size	Volum e	Rate	GPS ed	foto	License / Condition of use
Graph road	Energy	Static	In	Struct	Understood	Name Surname	Stakehol der ID	url	Simple	Push	Datex	Sensor	XML	2 variable s	10Byte	Every 10 minutes	Yes	URL.	Public as CC...
Parking	Graph	Real Time	Out	Non struct	Acquired	Email	Staff or not staff	Broker	Certificate	Pull	WS	Sens- Actuator	JSON	15 fields	1245 Kbyte	Sporadic, max 1000 times per day	No	IMG	Link to file
Consumption of energy	Mobility	Combined	In/out		Scheduled	Phone	Internal..		Etc.		REST	KPI	GeoJSON			Periodic	Kind		Private ...
	Transport	RT stream			Tested	Etc..					Custom	Personal Data	KMZ			2 per day	Insid e msg		Restricted to ...
	RT Messages			Operative						ODBC	Ext Srv	WFS				Static ...		Sensible data
					Failed						JDBC	IOT	WMS						GDPR aspects
					Not needed							Virtual Sensor	GTFS						
												GIS	db						
												Heatmap							
												Path, trajectory							
												Trend							
																		

Examples are provided per colum. Thus, the resulted raws may have not sense.

The status refers to the ingestion process.

IV.B.2. Design: Data Modelling

In traditional development processes the data modelling would go on the identification of classes, they relationships, class diagram, and the definition of tables for the SQL database, performing normalization, etc.

In the case of IoT/WoT modeling the Data Modelling phase consists in the definition data structures/models (Entity Models) used in the solutions. In Snap4City, the approach is grounded on Big Data and Digital Twin, coming from IoT/WoT domains, event driven, and scalability being big data. The approach passes from

formalization of Entity Models which are identical from the data ingestion and data storage. The Entity Models on storage are directly indexed into the storage without the need to defining for them a specific table structure and relationships among tables, neither to perform normalization to permit the efficient indexing. All the relationships among the entities and the Entity Models are modelled into the knowledge base, and the Smart City API can efficiently query them by using multiple filters via noSQL approach, transparent for the developer.

This phase has to answer at questions such as:

- Which information is present in these data, it fits the purpose?
- Which data models are used, standard, custom, etc.?
- Which data models would be produced?
- How can I get the data?
- Are those data private/public, which licensing?
- Is the licensing compatible with the purpose?
- Are those data ethically compliant?
- Do I need to create a DPIA for GDPR?
- Do I need to create an agreement?
- Etc.

IV.B.2.a- Data: High Level Types, HLT

Thumb Rules on Snap4City for High Level Type and data types in general.

When in Snap4City is talking about Modelling is referring to Defining and Modelling Entity Models

A large number of other data types are also managed as follows.

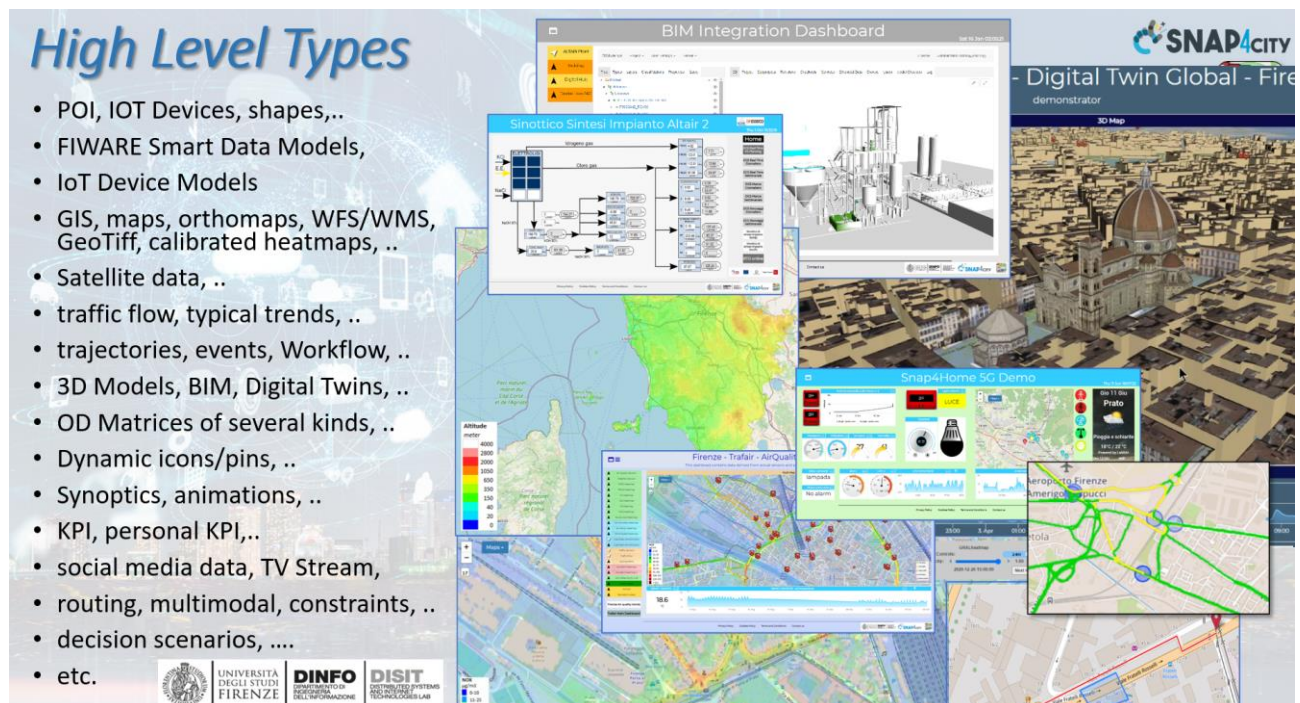


Figure 12: high level types, example.

Snap4City has the concept of **High-Level Type**, that are unified with a **Unified Data Model**. The HLT are the collection of all data kinds of data formats supported by Snap4City system and are listed from the **Data Inspector**, and in the **Dashboard Wizard**.

They are grouped by categories as follows:

- **Devices/Entities** and their model and variables (Managed from **IoT Directory**):
 - **Entity Model (IoT Device Model), Entity Instance (IoT Device), Entity Variable (IoT Device Variable)** (managed by IoT Directory)
 - Entity Models can be derived from FIWARE Smart Data Models
 - **Mobile Entity Model (Mobile Device Model), Mobile Entity Instance (Mobile Device), Mobile Entity Variable (Mobile Device Variable)** (managed by IoT Directory) The difference with the previous is the possibility for them to move (change GPS) and thus render them on trajectories, track for fleets and sensors, let's say time series moving over time.
 - Entity Models can be derived from FIWARE Smart Data Models
 - **Data Table Device, Data Table Model, Data Table Variable** (managed by IoT Directory and produce by Data Table tool). They are produced by the **Data Table Loader or POI Loader** automated processes for loading data and generating their models and devices, which automatically produce the model, devices, and instance messages over time, let say time series.
 - BIM devices are actually Entity Instances (managed by BIM Manager accessible into the **Dashboard Wizard**), that refer to some BIM View, and the BIM manager may refer to some BIM Views in which the Entity Instance.
 - **Files** devices are actually Entity Instances (managed by the File Manager from **Data Management**)
 - Sensor, Sensor Device, Sensor-Actuator (deprecated since all the IoT Devices can be both sensors and actuators). They are produced by formers ETL processes, similar to the concept of device and variable (Sensor Device, Sensor). In the present version of Snap4City, all sensors are actually modelled as sensors-actuators.
- **KPI, key performance indicator** single variables (Managed from **Data Management, HLT**):
 - **MyKPI** (managed by MyKPI manager from **Data Management**) they are all personal data produced by IoT App, Mobile App, or loaded from some process. MyKPIs are mainly used for trajectories, for saving single values time series. They can be: (i) used for long-term saved and retrieved from Proc.Logic / IoT App, (ii) saved on MySQL or Open Search according to a setting, (iii) used as event-driven for synoptics, (iv) created via the MyKPI management via the graphic user interface.
 - **My Personal Data, MyData** (the latter are mainly deprecated). The former are used to save user profile information such as password of hash, scenarios, etc.
 - **My POI** (mainly managed as Entity Models). They are static point of interest that can born private and then made public when approved. POI are used for marking Points of Interest on maps (museums, banks, schools, parks, etc.), they are points of attractions without time changing variables, without time series and with a relevant number of metadata: web page, email, time of opening, etc. They could be also generated as Devices, see for instance POI Loader. MyPOI are personal POI, which can be created from a Mobile App. The administrator can leverage a MyPOI to a general POI.
 - KPI: deprecated HLT
- **Map features** and tools in addition to all the above data which can be located on map (Managed from **Data Management, HLT**), also:
 - **Heatmap** (managed by Heatmap manager from **Data Management**) they are produced by some data analytics or directly loaded on the Heatmap Server. They can be shown on a map, and picket if loaded correspondingly. Some of the heatmaps can be computed on the client (non-calibrated), the others are generated to produce dense and regular maps calibrated according to a **Color**

Map standard (see **Color Map Manager**). The latter is distributed by the GeoServer via WMS/WFS protocol and can be animated in 24 hours if instances over time are present.

- **Traffic Flows** (managed by Traffic Flows manager from **Data Management**) they are typically produced from a traffic flow reconstruction algorithm to generate data in terms of Polyline JSON and loaded on Traffic Flow Manager which converts them into geo images for the GeoServer which distributes them via WFS/WMS. They can be time series of data to be used for showing animations daily of the traffic flow on the map.
- **Traffic Flow PolyLine**: they are produced from traffic flow reconstruction data in terms of Polyline JSON and can be used for rendering on 3D City Digital Twin as crests, on a 3D map. They can be time series of data.
- **OD Matrix** (managed by OD Matrix manager from **Data Management**). They are produced from some original data, census data, cell phone, data, mobile data, etc., and loaded in terms of OD data on the OD Manager/server for distribution on map. They can refer to different geo areas, grids, ace, regional administrative shapes, etc. They can be time series of data, inflow and outflows, etc.
- **Tools**: what-if, scenarios, etc. (as tools accessible into the **Dashboard Wizard**) They can be shown on the Multi Data Map on the dashboard and are
 - **Scenario**: for creating critical scenarios blocking a part of the city, multiple unconnected areas and for multiple time slots. Scenarios can be saved and shared with other operators.
 - **What-if**: for exploiting a scenario and deciding routing and solutions to be taken for private routing. The resulting cases can be saved to be shared with other operators.
- **WFS** (WMS, etc.), for connection to GIS for images, maps, orthomaps, shapes, POI, PIN, etc.
- **3D models of the City, Global Digital Twin**. They are generated with a suite of tools starting from Lidar, GIS shapes, facades images, orthomaps images, etc.
- **Trajectories**, Paths, shapes, etc. they are produced from some original data or some data analytics, or routing, or paths. They can be sent to the map for visualization. Some segments of the Trajectory can be requested to the composed by a new routing.

- **Third party html pages and tools, only for Dashboard Widgets**

- **External Service** (managed by the **External Service Manager**) they are websites which registered on Snap4City.org and can be rendered into an Iframe of the Dashboard if they allow to do it. Some restrictions may be applied by the owner of the Web Sites.
- **Synoptics** (managed by the Synoptics Manager from **Extra Dashboard Widgets**) they are special modules provided by Snap4City grounded on SVG format for rendering in real time graphics layouts of energy plat, industrial plants, metro stations, etc. They may present several IoT Device Variable, Constants and MyKPI in event-driven data. They can also provide interactive elements by which the user can send data, and actions to the platform, writing to MyKPI, using the IoT App. Synoptics are instantiated based on a model which can be generated according to guidelines.
- **MicroApplications** (managed by the MicroApplications Manager from **Extra Dashboard Widgets**) they are Web Applications developed by Snap4City as micro solutions, micro applications solving specific cases and rendered into an Iframe of the Dashboard.
- **BIM View** for single building and areas (managed by BIM manager from **Data Management**)
 - **BIM View**: are used to represent and show a 3D building, floor or segment derived from some IFC format for BIM. They are collected and provided by the **BIM Manager**.
 - **BIM Device**: are IoT Devices used to represent and show a 3D building, floor or segment with a specific IoT Device, ServiceURI, POI attached. They are derived from some IFC format for BIM, collected and provided by the BIM Manager, while the BIM editor permits to attach the ServiceURI to the BIM corresponding element.

- **TV Cam** (managed by **TV Cam Manager** from **Data Management**), they are a specialized Entity Model and Entity Instances. So that, they have GSP location, nature Subnature, variable, etc., plus credentials to access at the protected video stream. They get video streams on RTSP, ONVIF modelled as Entities / Devices. They can be defined and managed by the **TV Cam manager**, in which it is possible to define protocol and access data. The visualization is performed on **WebRTC protocol** via **Kurento** on a special **Widget**.
- **Other HLT and Widgets:**
 - **Dashboard-IOT App:** representing the data flow, even driven from Proc.Logic / IoT App to Dashboards and vice versa. They can be intercepted and exploited.
 - **Special Widget:** several special widgets have been created to show specific HLT, for example the
 - **weather forecast, civil protection,**
 - **quality of public transport,**
 - **status of a specific decision support system process,**
 - **first aid, clock,**
 - **Twitter Citations, and Twitter Hashtags.**
 - **Complex Event:** to represent events of different kinds on specific Dashboard Widgets: entertainment events, critical events, operator events, traffic accidents, etc. In the short future, a new version of the Event Device and Event Table Widget will be finalized to deprecate this former HLT with a set of more flexible solutions.

IV.B.2.b- *From Data Modeling to Entity Messages*

Thumb Rules on Snap4City from Data Modelling to Messages.

In Snap4City:

- The relationships among models, entities/devices and messages are depicted in **Figure 9**
- **the Entity Models (IoT Device Models)** can be simply **instantiated** from:
 - **FIWARE Smart Data Models**, versioning, and harvesting the standard repository (**from Entity Directory / IoT Directory tool of Snap4City**)
 - **Entity Model / IoT Device Model** which are accessible into the Snap4City environment (**from Entity Directory / IoT Directory tool of Snap4City**)
 - Excel files by using **Data Table tool**, which extracts the model from the excel table and automatically creates IoT Device Model, IoT Devices and data attached to them
 - Creating a **custom Entity Model / IoT Device Model** in standard Snap4City format (**from Entity Directory / IoT Directory tool of Snap4City**)
- **the Entity Models** can be customized.
- **Each Entity Models / IoT Data Model is a template** to produces **Entity Instances / IoT Devices** by registration (virtual or physical, does not matter). In the model creation one can specify static attributes describing
 - nature/sub-nature
 - a number of static variables, just defined once at the device creation.
 - if the device can move or not (be mobile device or not), changing GPS data
 - if the device is a certified one via Blockchain (beta version)
- **Each Entity Models / IoT Data Model can**
 - Be modified in terms of attributes, variable, etc., without creating any change in the Entities Instances / IoT devices already produced in the past from the model.
- The **Entity Instances / IoT Devices** can be classified into

- *Physical Entities* (**token, on board unit on the vehicle, device for assessing the drive attention, etc.**). All *Physical Entities* are sending data on the platform on server via some Broker or other data ingestion process, for example via some local application, third party service, Edge solution, etc.
- *Virtual Entities* (**Vehicle, Driver, DriverHealthiness, VehicleEvent, DriverEvent, DriverAnalysis**). *Virtual Entities* are managed on server side for managing data and providing a set of user interface views for data entry (registering data) and for data rendering visualization.
- **Entity Instances / IoT Devices can be created in Snap4City** by means of:
 - Manual production via **Entity Directory / IoT Directory**
 - Massive production via:
 - **Data Table tool of snap4City**
 - **Processing Logic / IoT App by using MicroService for**
 - **Entity Instance / IoT Device production from Model**
 - **Harvesting external Orion Brokers, also via Entity Directory / IoT Directory**
 - **API and MicroServices**
- **Entity Instances / IoT Devices**
 - Can be modified in terms of variables and variable types without affecting the Entity Model, neither the values of past data from Entity Messages.
 - Cannot be modified in term of Entity/device name which is unique forever.
 - Can be canceled, but actually the cancelation is not a physical delete of data and device, it is only a change of status on deleted, so that from administrator you can recover the case.
- **Entity Messages** are data messages sent over time changing the value of the Entity. They have to be compliant with the Entity Instance, and may create Time Series as described in **Figures 9 and 12b**.

IV.B.2.c- How to perform Data Modeling

How to proceed to the Entity Modeling / Data Modelling:

- **Define a formal table for each Entity Model**, eventual exploitation of available entity models from some library such as those of: Snap4City IoT Device Models, FIWARE Smart Data Models. Both of them can be specialized, or in alternative a new custom **Entity Model** has to be defined.
 - **Snap4City is GDPR compliant**, but a certain attention has to be applied in collecting and using private data. All the data saved into Snap4City are private, while to make them public is a click which the owner of the Entity Instance (Device) can perform!
 - on Snap4City platform all **data models** are coded as an **Entity Models / IoT Device Models**.
 - **Entity Models / Device Model** can be created, imported and exploited from Snap4City **Entity Directory / IoT Directory**
 - Experts of Snap4City may directly create the **Entity Models / IoT Device Model** on the **Entity Directory / IoT Directory tool**
- Create one or more **Entity Instances / IoT Devices** from a single **Entity Model / IoT Device Model**, by providing some information such as the Identification/name and other static variables (see example in **Figure 9**). In Snap4City, a large number of tools allow to
 - create Entity Instances / IoT Devices from scratch and from **Entity Model / IoT Device Model**
 - massively create huge number of Entity Instances / IoT Devices from an **Entity Model / Device Model** via process in Bulk from file, and from Processing Tool / IoT App
 - massively create huge number of Entity Instances / IoT Devices from Data Table tool which create also the **Entity Model / Device Model** learning from an excel file and from it the device and Entity Instance Message / data messages.
- **Any Entity Instance / IoT Device** may change status and value over time, thus it may receive a large number of **Entity Messages** attached to it (see example in **Figure 9**). Those messages can describe the **Entity Instance** evolution over time of a set of attributes including:
 - **state or versioning**. For example, modelling a user profile, could change over time: driving,

sleeping, vacation, etc.

- **variables or attributes (float, integer, etc.).** For example, for a device modelling a Car we could have at a certain time and date (so called dateObserved, with value type = timestamp): position (lat, lon), velocity, acceleration, engine status, number of cycles per minute, travelled km, etc.
- **references** to other Entity Instances / IoT Devices (in Snap4City, they coded as **ServiceURI**). For example, a Driver will have references to its own connected entities such as: DriverEvent, DriverHealthiness, and DriverAnalysis where information about tickets taken, analysis performed, changes of the driver license will be stored. Similarly, a Vehicle will have references to a VehicleEvent, with records for the maintenance of the vehicles, etc.
- **see:** <https://www.snap4city.org/download/video/course/iot/>
- **see also:** <https://www.snap4city.org/download/video/course/di/>

Please note that, in all the **Snap4City Entity Models (IoT Device Models)**: Nature and Subnature (classification), Lat/Lon (GPS positioning), and Device in Mobility (yes or not) attributes (variable) of Entities/Devices are not replicated in all the other models, since they are defined for default for all the Entity Models and for all the Smart Data Models as well. In **Figure 12a**, the differences from Entity Model (IoT Device Model), one of its instances the registered Entity Instance (IoT device) “park45” and a specific message over time for the park45 device are put in evidence. Please note that for:

- **Entity Model (IoT Device Model):** black strings are reporting model definition, while in red the set parameters for the model. In some cases, the model may not impose the protocol and broker.
- **Entity Instance (IoT Device):** in instantiating the part45 device from its Model, a number of parameters in red has been imposed.
- **Entity Message (A temporal instance):** a message over time (dateObserved, with value type = timestamp) and at a given location (GPS position) has been sent to the Entity Instance park45, also providing a set of changed values for the attributes/variables: FeeSlots, Humidity, Temperature, etc.

Where	Entity Model (IoT Device Model)	Entity Instance (IoT Device)	Entity Message a Temporal Instance
Broker	Broker: OrionUNIFI		
Broker	Protocol: NGSI		
Info	ID: string	ID: “ park45 ”	park45
Position	GPS: lat, long	GSP Position: 43.12, 11.34	GSP Position: 44.12, 11.12
Static attribute	Description: string	Description: “ parking massaia ”	
Static attribute	Location: string	Location: “ Via Massaia ”	
Static attribute	Civic Number: string	Civic Number: 3	
Static attribute	MaxCapacity: number, cars	MaxCapacity: 456	
Values	dateObserved: Timestamp		23-12-2019T20:13:12...
Values	FreeSlots: Integer, #		345
Values	Humidity: float, %		25,5
Values	Temperature: float, celsius		34

Figure 12a: Example of Entity Model (IoT Device Model), Entity Instance (IoT Device) and an Entity Message as temporal instance

- **create Data Model Diagrams** which describes the relationships among Data/Entities, and in particular on Snap4City, among the Entities Instances and Entities Messages (see **Figure 13, right**).
 - Connecting different kind of Entities Instances/Messages via ServiceURI, SURI: in fact, the actual connections can be among Entity Instances and/or Entities Messages. Thus, creating a sort of Interaction Diagram of UML. Connections among Entities can be:
 - from a Static Attribute of an Entity Instance to another Entity Instance, as highlighted in

green in **Figure 12b**.

- from a Value/Variable of an Entity Message of an Entity Instance to another Entity Instance, as highlighted in green in **Figure 12b**.

Where	Entity Model (IoT Device Model)	Entity Instance (IoT Device)	Entity Message at 23-12-2019T20:15:00	Entity Message at 23-12-2019T20:30:12
Broker	Broker: OrionUNIFI			
Broker	Protocol: NGSI			
Info	ID: string	ID: "park45"	park45	park45
Position	GPS: lat, long	GSP: 43.12, 11.34	GSP: 44.1256, 11.1234	GSP: 44.1259, 11.1233
Static attribute	Description: string	Description: "parking massaia"		
Static attribute	MyAddInfoSURI: string	MyAddInfoSURI: "http://...../InfoPersonal"		
Values	dateObserved: Timestamp		23-12-2019T20:15:00	23-12-2019T20:30:12
Values	FreeSlots: Integer, #		FreeSlots: 345	FreeSlots: 234
Values	TodayCarSURI: string		TodayCarSURI: "http://...../CarNF126GD"	TodayCarSURI: "http://...../CarGF789KK"
Values	Temperature: float, celsius		34	34

Figure 12b: Connections among Entities Instances and Message: the example reports a static connection and dynamic connection to change the car at a given timestamp. Please note also change of position and other parameters, if needed

- Different kinds of relationships are possible depending on the Data Type used into the Entity Model (IoT Device Model), for example,
 - Single **ServiceURI**
 - Vector of **ServiceURI** (used only when simultaneous events occur, for example, list of changes performed to an Engine)
- To each **ServiceURI**, it is possible to refer to any kind of Entity Instances (IoT Devices, Mobile Devices, POI, etc.), it is a sort of pointer/reference. Once obtained the reference it is possible to ask at the Entity to provide its last values (as well as the value at a given dateObserved, with value type = timestamp), and thus accessing to its definition, a sort of reflection model of high-level programming paradigms.
 - For example, this means that it is possible to have attributes as ServiceURI and attaching to it some Entity Instance and its Messages for describing the occurred event:
 - Analysis, Maintenance, Ticket, Illness, Vacation, etc.
- Optionally: if you like you can create **Entity Relationships** diagrams among Entities, see UML, as a plus.

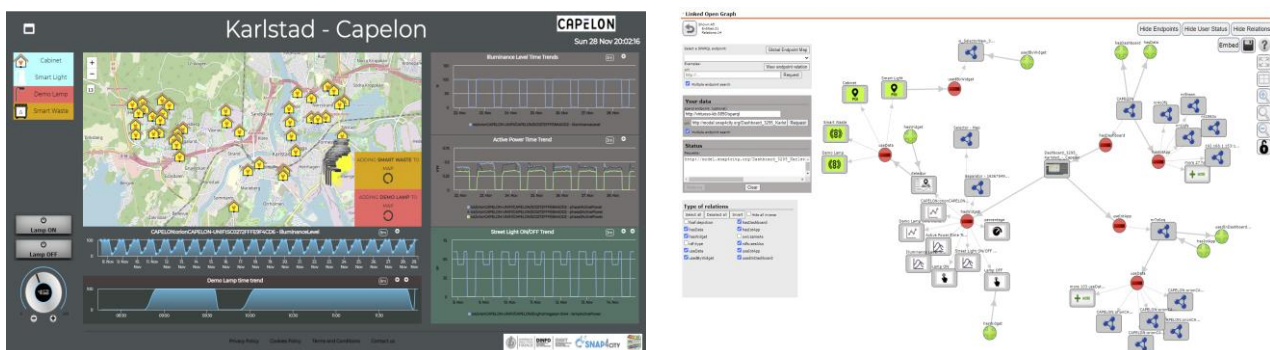


Figure 13: Example of a Dashboard and its graphical representation of the relationships among widgets, devices, Entity Instances and Entity Messages, etc. The tool on the right side is called LOG, Linked Open Graph and it is accessible from the dialog of Dashboard manager.

The **Snap4City** tools for supporting the activity of Data Modelling are:

- **Semantic Reasoner** (accessible as **ServiceMap/KnowledgeBase** and **SuperServiceMap**) supporting the modelling of city entities in multiple and federate knowledge base, their semantic relationships and search, for they discovery, Entity/IoT Discovery, dynamic and autonomous data analytics as machine learning, and for development of applications in multiple domains: mobility and transport, tourism, health, welfare, social, etc. Extending the Km4City multi-domain semantic model (<https://www.km4city.org>); Federation of Smart Cities via **smart city API** for large horizontal scaling and world scale coverage without expensive solutions.
- **Entity Directory (IoT Directory)** and service simplifying the creation of applications abstracting complexity of data models, Entity Instances, IoT Devices, IoT Edge, Brokers, IoT Brokers, protocols and data formats, exploiting the Semantic Reasoner, addressing FIWARE Smart Data Models, Snap4City Entity models (IoT Device Models), and any custom data model you may have; attaching in a few minutes External Brokers and registering their corresponding Entity Instances / IoT Devices into the platform, and thus into the Knowledge Base.
- **Open-Source code** for implementing and connecting IoT Devices and IoT Edge on Android, Arduino, Raspberry Pi, Linux, Windows, ESP32, Arm, AXIS cameras, etc., with mutual authentication and encrypted communication, and Processing Logic / IOT App control from web page.
- **All the High-Level Types can be created and manipulated by Processing Logic / IoT App and may be rendered on Dashboards from their corresponding Data Managers of the Data Management.**

IV.B.2.d- Concept of ServiceURI and HLT Identifiers

ServiceURI is the identifier of an Entity into the Knowledge Base and thus also of all the Entity Instances. References are established via the so-called **ServiceURI** in the Knowledge Base to refer to: IoT Devices, Entity Instances, POI, etc.

[illegible]

Please note that the **ServiceURI** of a given data model / Device / city entity / knowledge base entity can be recovered from a number of tools: Data Inspector (from healthiness button, red/green), ServiceMap, SuperServiceMap, Dashboard Wizard (from healthiness button, red/green), IoT Directory.

In Snap4City, **ServiceURI**, also called **SURI**, is defined a Snap4City Entity/Service URI (also called in IoT Directory Device URI) into some Km4City Knowledge Base.

For example as: <http://www.disit.org/km4city/resource/iot/orionUNIFI/DISIT/METRO759>

ServiceURI is an URI identifier defined according to the standard:

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

If you put the SURI on a browser, and the SURI refers to a public entity, its definition appears on browser as in the figure aside, since it is compliant with the Linked Data standard. You can see how the entities are identified in terms of URIs.

Please note that:

- The identifiers of other HLT are not based on SURI, for example for:
 - MyKPI the IDs are in the form of: 176986678846/Numero
 - Heatmaps the IDs are in the form of: 15MinIndex_CultureAndCultsIndex
 - etc.
- **consult the Data Inspector** on the platform to identify the corresponding shape to identify each kind of HLT. The identifiers are used by the
 - corresponding managers: MyKPI Manager, HeatMap manager, OD Manager, etc. Progressively, all those HLT will go to be referred also by using a SURI.
 - Dashboard manager and Wizard to connect the HLT to the specific widgets.
 - Referring to them as a response of some request on APIs and thus on queries.

IV.B.2.e- Concept of Time Series

The **Time Series** are sequences of data values with an associated time stamp. In Snap4City, any Entity and MyKPI can be used to model a time series. MyKPIs provides only a single variable changing over time, and thus are limited, also since, so far, they are not modeled into the KB. They started as personal information to be put in safe.

In Snap4City, a more diffusely used and powerful solution for Time Series is based on Entity Instances which may have a number of variables, each of which can change over time, thus creating a Time Series. Therefore, an Entity Instance (IoT Device) may send a new measure with its related time stamp (date and time) (Red Lines) in **Figure 14**. This means that regular or unregular time slots can be valid as well.

For each Entity Instance, new measures (Entity Messages, as in Figure 12b) are conventionally time ordered according to variable «dateObserved», with value type = timestamp, which has to be Unique into the Entity Model definition.

If other variables/values need to keep a timestamp, they have to be called in other manner and have to be of value type “DateTime”.

The example of **Figure 14** reports a device with two time series, humidity and temperature. The advantage of having the data variables organized as time series is that they can be represented in a multiple and single time trend graph diagrams. The example of **Figure 9** reports a device user45driveanalysis with multiple timeseries: location, doctor, status, etc. **Figure 12b** reports the evolution of an Entity Instance via its Entity Messages.

Entity / IOT Device

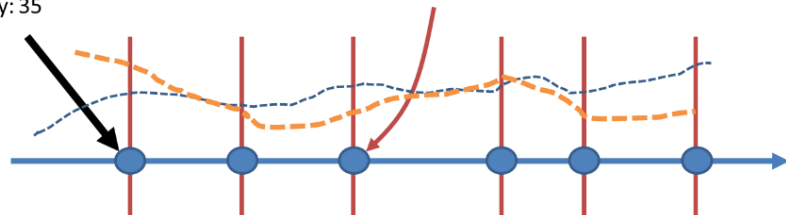


Entity: IOT Device

Sends a message

Message (timestamp: 02-04-2020 at 10:30, Temperature: 29.34, Humidity: 35)

dateObserved	Temp	Humidity
02-04-2020 10:30	34.5	23
02-04-2020 10:40	36.5	24
02-04-2020 10:50	36.0	22.5



- A set of data coming from an Entity Instance / IoT Device with multiple sensor become a time series of values for devices.
 - For example: taking a new measure every 10 minutes (**Red Lines**)
 - Non regular rates can be valid data as well.
- Each new measure in Snap4City is conventionally time located in «**dateObserved**», which has to be **Unique**.
 - **Only one message per dateObserved is allowed**

Figure 14: Entity Instance (IoT Device) and Time Series from its variables.

Please note that, in all the **Snap4City Entity Models (IoT Device Models)**:

- Nature and Subnature (classification), Lat/Lon (GPS positioning), and Entity/Device in Mobility (yes or not) attributes do not need to be specified by the model designer since they are defined for default for all the Entity Models (IoT Device Models) and for all the Smart Data Models as well.

On the contrary, if you would create an Entity Model producing **Time Series** of the Entity Instance (thus accepting Entity Messages over time, you have to:

- Define a variable named **dateObserved**, with value type = timestamp, for the Entity Instance (IoT Device).

If the dateObserved, with value type = timestamp, is not specified, the versioning or the state evolution are not time variables, so that the device is substantially static, and its variable cannot be taken as time series. At each timestamp, a new Entity Message can be posted for a given Entity Instance. Thus, the Message will provide a different value for the dateObserved.

Please note that, values assigned to dateObserved in Entity Messages can be in past for loading historical data, and in the future for loading predictions, future data over the time series.

Moreover, at each new timestamp/Message of a given Entity Instance (IoT Device), the other variables (e.g., V1, V2, V3, etc., of **Figure 15**) can be updated/provided or not. If they are not updated/received, there are two options:

- omitting the new value,
- putting a null as a value.

The two cases are producing different effects on rendering tables of those data on Dashboards and on time trends (multiseries graphs) as reported in **Figure 15** and explained in the following.

The messages posted on Entity Instances / IoT Devices can produce different effects on time series.

Omitting the message would allow the broker to reuse the last data to fill it, as for V5, which appear

- valid in all messages on graphs
- With holes in tables

Putting null values would produce a missing data and thus would lead to create:

- interpolate line on graphs: dashed are actually continuous lines in Dashboards
- Empty values in the tables

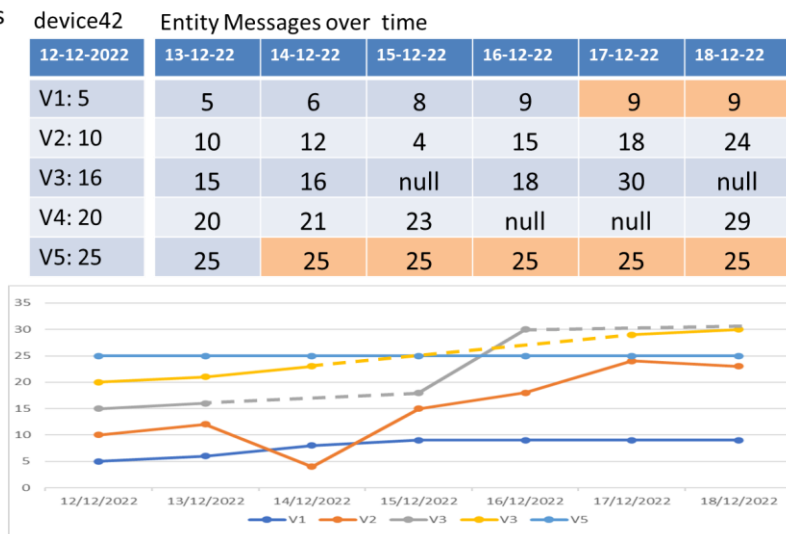


Figure 15: tricks and tips on time series.

The counter indication of omitting the variable value can be very relevant.

(a) omitting the new value

If the data values are not provided, the broker is keeping its **last value** and thus this last value is saved into the platform storage as well (see **Figure 15, cases of V1 and V5**). Thus, the values shown on graph (in this case in dashed line, while on dashboard will be continuous), is the last value. On the other hand, it can be very old, it come from the last measure, may be of 1 months ago. In some cases, this approach is viable. For the rendering on table (the value is present in the storage), so that the data replicated will be shown and thus the orange cells will not be distinguishable from the others since they have been feed into the storage. If the above approach is not acceptable, one has to write "null" in variable value thus going for case (b) as follows.

(b) putting a null as a value

Putting in the Entity Message a null value for certain variable will imply to write it into the broker and the broker will not post any value on the storage, leaving a hole, that will be interpolated in the graphs if a new value will come later, otherwise not. The graph can be inspected with mouse to pop up the actual values and where they are, using the dots, where not dots are there means no data. Moreover, the Tables are going to show empty cells where there are the "null"s since the data is missing. The same effect of null in requesting last values may happen in rendering data on barseries, spidernet, single content, donut, etc. It may be corrected to send them values via some Business Logic which could enlarge the request to the Recent Last value, for example with a time interval comparable to the rate of the device or two time of the device rate, etc. This logic has to be implemented by case, by the developer on Client or Server Sides Business Logics.

These approaches are also impacting the queries requesting the last value of an Entity Instance (IoT Device) from the Smart City API and from the broker as well. An alternative approach may be to split the Entity Model in smaller Models having variables which are sampled at the same time, sent to the broker and thus written at the same time, at the same dateObserved. The smaller models can be also referred to an aggregating Model which links them via their ServiceURIs, and vice versa if needed, etc.

IV.B.2.f- Rule for Entity Models/Instances, names and values

In the definition of an Entity Model (IoT Device Model) for each variable you have to define the **Variable Name** and for it the **Value Type**, **Value Unit** and **Data Type**, and for this purpose you have to use the Dictionary

reported in Section VII Appendix Data Dictionary. The Entity Directory (IoT Directory) has an on-line Dictionary enforced into the assisted tool for defining the Entity Models (IoT Device Models), and for the definition of full custom Entity Instances (IoT Devices). An online version of the table in **appendix** can be recovered from <https://www.snap4city.org/818> This online version is not continuously updated, while the one enforced into the Entity Dictionary is updated in real time.

Any update and addition to the dictionary of snap4city.org has to be requested emailing to snap4city@disit.org of just for some help in their definition.

If you have your own instance of the platform, you can define your own dictionary and request a copy of the snap4city.org dictionary by email.

Please note that Entity Models/Instances have to follow the rules:

- **ValueName** is the name of the variable/values,
 - it does not accept any space or special char in its definition as in most of the programming languages.
 - for each Value Name you have to specify **Value_Type**, **Value_Unit** and **Data_Type**,
 - **Value Name cannot be one of the strings used for Value_Type, Value_Unit and Data_Type**
 - **see** Section VII Appendix Data Dictionary.
- according to the **Value_Type** selected the possible values for
 - **Value_Unit** would be restricted. These constraints are directly applied in the user interface tool of the Entity Directory (IoT Directory).
 - **Data_Type** would be restricted. These constraints are directly applied in the user interface tool of the Entity Directory (IoT Directory).
- **Value_Type**, **Value_Unit**, **Data_Type** cannot have spaces in their name definitions.
- **Data_Type** for **Value_Unit** defined as timestamps are strings. They are expected to be formatted as ISOString: YYYY-MM-DDTHH:mm:ss.sssZ which can be obtained by using function toISOString() of JavaScript on variables of kind Date()
 - Please note that ISOString have to be in UTC to avoid misplacement of events coming in different time regions in the wrong position, and thus to reproduce wrong visualization once you change the international time fuse.
 - Some examples:
 - "2020-08-04T04:00:00+02:00",
 - "2020-08-03T00:00:00.000Z"
 - Some code to get it:
 - ```
var strnow = new Date().toISOString();
```
  - for fuse management

```
var todaynow = new Date();
dateCET2Z(todaynow).toISOString();

function dateCET2Z(date) {
 d = new Date(date).toLocaleString('nl-BE', {timeZone: 'Europe/Brussels'});
 offset = new Date(d).getTime() - new Date(date).getTime();
 return new Date(new Date(date).getTime() - offset);
}
```
- Time **Durations** can be in milliseconds, hours, minutes, etc... and are typically integer or float.
- **Each Entity Model and Entity Instance may have**
  - **At most one** ValueName defined as **dateObserved**, with value\_type = timestamp.
  - **At most one** Value Type defined as Timestamp
  - **dateObserved** should have **Value\_Type** defined as **Timestamp** to create Time Series

- any other variable, and thus Value Name which needs to have as Value\_Unit of timestamp in millisecond has to provide a **Value\_Type** defined as **Datetime** which is also coded in **time stamp millisecond** and in **ISO string**.
- The **absence of unique single Value\_Type per model/entity** defined as **Timestamp** would bring to create a stable Entity without time series.
- The **presence of multiple Value\_Type per model/entity** defined as **Timestamp** would bring to create unpredictable behavior.
- **forbidden** ValueNames are: type, id, value;
- **Data\_Types** are typically:
  - **integer, float (AKA numeric)**. Any variable on which you would like to apply math operators on queries (and thus on the corresponding Smart City API) such as ==, <, >, <=, >=, etc. has to be defined with a Data\_Type which is numeric. Integer and Float do not have limitations on their dynamics.
  - **String**. Any variable can be also defined and loaded with strings, and you can send on strings also numbers. The only operator on strings is the verification of the equality ==. Good for status detection.
  - **JSON**. Can be JSON data structure.
- **JSON Data\_type** have some limitations. Since if your have **JSON data type**:
  - The loaded JSON pack will be stored as a string into the Storage
  - The loaded JSON data will not be
    - singularly indexed into the storage, so that they will not be automatically usable into Time Series, and dashboards for showing values, sequences, barseries, spidernet, etc.
    - search-able by using queries and query by value of the Smart City API

#### IV.B.2.g- Examples of Data Models

In this section, examples of data models related to the example/solution reported above for monitoring Vehicles and Drivers are shown. Entity Models (IoT Device Models) for Drivers, driverHealthiness, Vehicle, and VehicleEvent, etc., are described.

| Entity Model / IoT Device Model: ..... |            |                 |           |
|----------------------------------------|------------|-----------------|-----------|
| Nature:.....                           |            |                 |           |
| Subnature: .....                       |            |                 |           |
| Lat,lon: .....                         |            |                 |           |
| Device/Entity in Mobility: .....       |            |                 |           |
| Value_name                             | Value Type | Value Unit      | Data Type |
| dateObserved                           | Timestamp  | Timestamp in ms | String    |
|                                        |            |                 |           |
|                                        |            |                 |           |

Each model is represented as a table as with the following schemas.

##### Typical Entity Model (IOT Device Model) for User Profile

According to the above description, for each Entity Model that would be used to create Entity Instances with Entity Messages changing over time the dateObserved variable has to be defined. The dateObserved is the date and time at which the user profile is updated, not the timestamp of its insertion, but the timestamp of acquired data on the field. In the case of User profile, the evolving over time is particularly useful for versioning the user profile data and for tracking the general user profile state evolution. For example, to state that a driver is active or nonactive, etc., to save the time in which the driver changes the car, to save the time and data related to driver conditions and locations, to mark the timestamp and maintenance info of an equipment, etc.

|                                         |  |  |  |
|-----------------------------------------|--|--|--|
| Entity Model / IoT Device Model: Driver |  |  |  |
| Nature:.....                            |  |  |  |
| Subnature: .....                        |  |  |  |

**Lat,lon:** Default (they do not need to be specified in the variables, they are provided by default, but values have to be imposed at the instantiation of the device from model), they are float

**Device/Entity in Mobility:** No (the variable does not need to be specified, while the value has to be set to state if the Lat, Lon are going to change, moving the device or not)

| Value_name                       | Value Type       | Value Unit             | Data Type     |
|----------------------------------|------------------|------------------------|---------------|
| <b>dateObserved</b>              | <b>Timestamp</b> | <b>Timestamp in ms</b> | <b>String</b> |
| Identifier<br>(nickname for ex.) | ID               | text                   | String        |
| name                             | entity           | text                   | String        |
| surname                          | entity           | text                   | String        |
| age                              | age              | number                 | Integer       |
| sex                              | status           | some coded status      | String        |
| language                         | entity           | text                   | String        |
| email                            | entity           | text                   | String        |
| phone                            | entity           | text                   | String        |
| address                          | entity           | text                   | String        |
| locality                         | entity           | text                   | String        |
| city                             | entity           | text                   | String        |
| nationality                      | entity           | text                   | String        |
| civicNumber                      | entity           | text                   | String        |
| dateofBorn                       | DateTime         | Timestamp in ms        | String        |
| gender                           | status           | some coded status      | String        |
| driverHelthinessID               | Identifier       | ServiceURI             | String        |
| driverEventID                    | Identifier       | ServiceURI             | String        |
| driverAnalysisID                 | Identifier       | ServiceURI             | String        |
| VehicleID                        | Identifier       | ServiceURI             | String        |

#### Entity Model / IoT Device Model: driverHelthiness

**Nature:**.....

**Subnature:** .....

**Lat,lon:** .....

**Device/Entity in Mobility:** .....

| Value_name            | Value Type       | Value Unit             | Data Type     |
|-----------------------|------------------|------------------------|---------------|
| <b>dateObserved</b>   | <b>Timestamp</b> | <b>Timestamp in ms</b> | <b>String</b> |
| kind                  |                  |                        |               |
| levelAttentionFactor1 |                  |                        |               |
| levelAttentionFactor2 |                  |                        |               |
|                       |                  |                        |               |
|                       |                  |                        |               |
| driverID              | Identifier       | ServiceURI             | String        |

#### Entity Model / IoT Device Model: Vehicle

**Nature:**.....

**Subnature:** .....

**Lat,lon:** .....

**Device / Entity in Mobility:** .....

| Value_name          | Value Type       | Value Unit             | Data Type     |
|---------------------|------------------|------------------------|---------------|
| <b>dateObserved</b> | <b>Timestamp</b> | <b>Timestamp in ms</b> | <b>String</b> |
| producer            | entity           | text                   | String        |
| model               | entity           | text                   | String        |
| plate               | entity           | text                   | String        |
| companyID           | entity           | text                   | String        |
| velocity            | velocity         | km/h                   | float         |
| acceleration        | acceleration     | m/s <sup>2</sup>       | float         |
| Status              | status           | some coded status      | String        |
| energyLevel         | energy level     | percentage             | Float         |
| kmTotal             | distance         | km                     | Float         |
| thankLevel          | energy level     | percentage             | Float         |
| vehicleEventID      | Identifier       | ServiceURI             | String        |
| driverID            | Identifier       | ServiceURI             | String        |

#### Entity Model / IoT Device Model: VehicleEvent

**Nature:**.....

**Subnature:** .....

**Lat,lon:** .....

| Device / Entity in Mobility: ..... |            |                   |           |
|------------------------------------|------------|-------------------|-----------|
| Value_name                         | Value Type | Value Unit        | Data Type |
| dateObserved                       | Timestamp  | Timestamp in ms   | String    |
| eventID                            | ID         | text              | String    |
| eventKind                          | status     | some coded status | String    |
| status                             | status     | some coded status | String    |
| vehicleID                          | Identifier | ServiceURI        | String    |

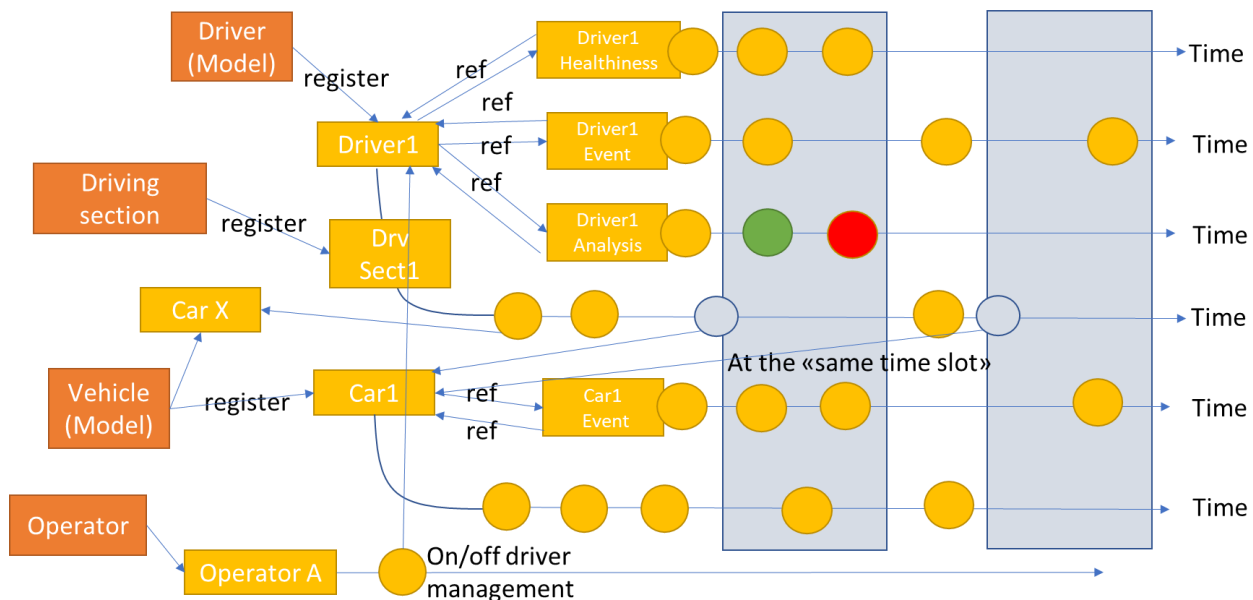
For example, at level of their relationships we have the example of Figure 16.

In Figure 16, an example of the above provided Entity Models:

- We can assume that each User playing the role of Driver (or operator) would be registered on Snap4City as Manager role into some organization indicated by the platform manager.
  - Once registered the User is going to provide its own/preferred **Nickname** (also email, that is the username) and password to access at the platform on web and mobile App.
  - The Mobile App and Web app should be compliant with the Authentication model described in the following.
- The **doctor/operator** would enter the information of each User involved into the experiments. Thus, the operator will perform the registration of the Device / Entity representing the User and connect the Entity with the NickName (user name on platform). To this end a dedicated Dashboard for registration is produced that will provide forms to be filled and the Proc.Logic / IoT App to register a Device for the User by using the device from model node in node-red.
  - The **Dashboard for registration** should be designed to request via a form the User's personal data according to the above *user profile Entity Model*, and thus including the platform **Nickname into the Proc.Logic / IoT App**.
  - To collect data from a dashboard it is possible to use at least two solutions from Proc.Logic / IoT App:
    - Use FORM node of dashboard widgets: <https://www.snap4city.org/714>
    - Produce a form on HTML page, and export into some External Content. <https://www.snap4city.org/618>
- The **Dashboard for registration** will have a Processing Logic (IoT App) receiving this information and:
  - registering Entity Instance from Entity Models:
    - Driver1 from Driver Model
    - Driver1Healthiness from DriverHealthiness Model
    - Driver1Event from DriverEvent Model
    - Driver1Analysis from DriverAnalysis Model
  - write into the Driver1 Entity Instance the user nickname to establish a link.
  - write the first messages for devices: Drive1, Driver1Healthiness, Driver1Event, Driver1Analysis to establish the reciprocal relationships via SURI.
  - delegate in reading to the User Nickname the devices: Driver1, Driver1Healthiness, Driver1Event, Driver1Analysis.
  - perform delegations in reading to enable doctors, analysts, etc. it can be possible to have a group of users (group of doctors, group of operators, group of analysts), and thus to make those delegation to the group. All platform user joining the group will be automatically authorized.
- The User can access to the Entity Instances in reading and thus can read data from its mobile app once authenticated.
- Conceptually, the data and the results of the analysis will be produced by some operator/doctor and thus not from the User.
- If the User or some its device when authenticated, it has to send data to some device, and thus it has to be the owner or to be delegated in writing for that device. The User may need to produce data coming from some device such as mobile phone tracking its position, etc. In this case, he has to be the owner of

the device, or to be delegated in writing (READ\_WRITE), according to the GDPR he should be capable also to cancel or overwrite the data, etc. A workaround could be to receive the data on platform permitting at the user to write the device. Periodically copy those data in a shadow device owned by the organization which is guarantee the long terms safe of the data.

- May be with some blockchain (work is in progress on the blockchain feature on Snap4City).



**Figure 16: Example of relationships among Entity Models (IoT Device Models), Entity Instances (Devices) produced from them, and Entity Messages (device messages) posted for Entity Instances (devices). In the picture, it has been added an entity for modeling the Drive Section (DrvSect1). It describes the section by means of its start and end of the section and the vehicle used (other info can be added as well, such as root, motivations, delivering info, etc.). In addition, the events can be included into the drive section or outside. Moreover, also the operator which may manage a set of possible Drivers has been added. For example, by recording any start and end of established relationship from the Manager and each driver, etc.**

**Please note that:** since the names of the Entity Instances involved are all declined with the same ServiceURI prefix, the Mobile App needs only to know the Driver SURI to know the SURI of the connected Entity Instances. On the other hand, a more formal approach would be to save the SURI of the connected Entities directly into the Driver Entity Instance by using a Entity Message, in fact some SURI may change over time, such the car which is going to change driver every day, for example.

See section IV.C.1.d to see how to maintain aligned different devices with a sort of Trigger / Sync flow implemented in Proc.Logic.

The advantage to have reported all the data in time series would be evident in the access to the data and in the rendering of the data on the graphs, including tracking paths on maps for moving devices with GPS.

#### IV.B.2.h- How to Create Entity Instances / IoT Devices, and their messages

In Snap4City, the creation of Entity Instances / IoT Devices can be performed by using several different approaches via:

- **Entity/IoT Directory**, which provides a user interface for REGISTERING Entity Instances / IoT Devices creation as:



- a full entity creation from scratch.
- an Instance of the Entity Model: Smart Data Models of FIWARE, IoT Device Models, Smart Models, Entity Models.
- a process of Harvesting of an External Broker which are Orion Broker
- **Proc.Logic / IoT App** by using a specific Node-RED node (Snap4City Library) which allows to create one or more Entity Instances / IoT Devices
  - from a known Model: Smart Data Models of FIWARE, IoT Device Models, Smart Models, Entity Models.
  - at the arrival or discovery of new Entity on the basis of a Model.
- **Data Table Loader**, which allows to load and excel file which is interpreted to create Entity Instances / IoT Devices, and also messages according to the Excel kind and data.
- Python **FastDataLoader** tool for automating the creation of Entity Instances / IoT Devices together with some data ingestion process.
- Specific **Entity/IoT Directory API**, which are those used by Python or other programming languages.

The above solutions are described in:

- <https://www.snap4city.org/download/video/course/iot/>
- <https://www.snap4city.org/download/video/course/di/>

Once create the Entity Instances the Entity Messages can arrive via:

- Brokers, in push.
- Proc.Logic: in push or pull.
- Python, or other custom process: in push or pull.

#### IV.B.2.i- *High Level Type vs Storage and distribution channel*

In Snap4City, the arrival of a new entity/data model does not imply for the developer to adequate the storage structure, but to model it on the Entity/IoT Directory once. Once a new Entity Model is registered, any instance can be directly deployed, and specific data loaded. Entities/Data entering in the platform can be automatically (i) stored (in **Figure 3**: Open Search), (ii) semantically indexed, (iii) directly sent to the user interface, or to data analytics for stream processing. The same data flows work as data driven coming from dashboards to data streams for acting and sending commands, alarms, etc., in the platform and/or to external devices via brokers or processes. Data stored (in some cases, the storage is called data shadow) can be browsed by using the Entity Inspector/Manager, for all the entity/data types managed in the platform. From simple data entities to IoT devices, maps, heatmaps, flow, matrices, 3D interactive representations, scenarios, etc., as needed by the DT.

**The Big Data Storage** is managed by multiple solutions:

- **Knowledge Base**, KB, (based on Km4City ontology) implemented as an RDF store (Virtuoso) which is an index for geospatial, relational, and temporal aspects. The KB can be federated, for example for federating different KB or different organizations. Whenever a new data model (Entity Instance or Model) is registered in the system, the registration is performed into the KB. Different instances of the KB can be federated via Smart City API by creating a mutual connection among cities/areas of the network, if the single installation wants to share, and deciding what to share.
- **Time series** from Entity Instances / IoT Devices and MyKPI are automatically feed into the storage Open Search cluster (which is the new name of Open Distro for Elastic Search of AWS) for storing and indexing data.

- The Entity Instances / IoT Devices need to be registered into Entity Directory / IoT Directory, which in turn automatically registers them on Knowledge Base for reasoning.
- The MyKPI can be also stored into some SQL database.
- **Heatmaps, Orthomaps, Traffic Flows Maps, and Maps** are managed by the Heatmap Server which is a GIS (for example GeoServer or ArcGIS if you have one installed with WMS facilities) and can distribute the maps according to WMS/WFS protocols towards Web and Mobiles App and Dashboards. They can be loaded from Processing Logic / IoT App using specific API.
- **Origin Destination Matrices** are managed by the OD/ODM Manager and represented into the Multi Data Map widget in dashboards. They can be loaded from Processing Logic / IoT App using specific API.
- **Internal Buildings 3D Shapes, and Floors** are managed and distributed by BIM Server. Standard BIM tools are used for editing and interchange in IFC formats with standard tools such as AutoDesk Revit, etc. Buildings and Floors are shown in dashboards for their integration with maps and time trends of IoT devices. The BIM are managed on Snap4City by the BIM Manager and can be loaded from the user interface using IFC format.
- **3D City Representation** can be loaded according to different components from which they are composed such as: image patterns, building shapes, 3D shapes, LIDAR data, traffic flows, POI, IoT Devices, etc. [DigitalTwin1], [DigitalTwin2], [GeneratingDigitalTwin]. They need to be loaded in agreement with the snap4city platform manager into the Dashboard area.

The Storage of Snap4City is organized as follows:

| High Level Type data               | Ingestion kind: MicroServices, API            | Storage                             | Distribution tool API                   |
|------------------------------------|-----------------------------------------------|-------------------------------------|-----------------------------------------|
| Entity / IoT Devices (time series) | Proc.Logic / IoT App, special Python, Brokers | OpenSearch, KB                      | Smart City API, Orion Broker            |
| MyKPI (time series)                | Proc.Logic / IoT App                          | OpenSearch (or SQL)                 | Smart City API                          |
| POI                                | Proc.Logic / IoT App                          | OpenSearch, KB                      | Smart City API                          |
| Heatmaps (time series)             | Proc.Logic / IoT App, Python,                 | GeoServer                           | API for WFS, WMS                        |
| Maps                               | API and user interface                        | GeoServer                           | API for WFS, WMS                        |
| GIS data                           | API and user interface                        | GeoServer                           | API for WFS, WMS                        |
| Traffic Flow (time series)         | Proc.Logic / IoT App, Rstudio, Python         | GeoServer, Traffic Flow Manager, KB | Traffic Flow Manager, API for WFS, WMS  |
| OD Matrices (time series)          | Proc.Logic / IoT App, Python                  | GeoServer, OD Manager               | OD Manager, API for WFS, WMS            |
| Trajectories                       | Proc.Logic / IoT App, Python                  | OpenSearch, KB                      | Smart City API                          |
| TV CAM                             | Registering and IoT Device                    | OpenSearch, KB                      | Smart City API                          |
| BIM view                           | BIM Server, IFC                               | BIM Server, KB<br>BIM Manager       | BIM Manager                             |
| 3D City Model                      | Python                                        | GeoServer, OpenSearch, KB           | Smart City API, API for WFS, WMS; HTTPS |

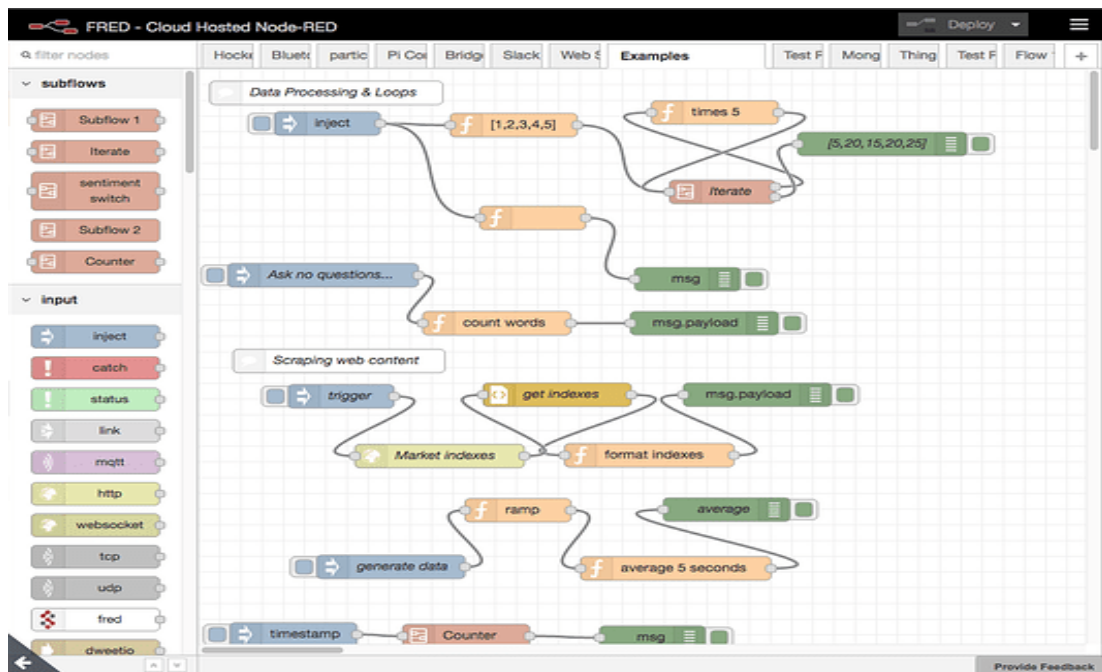
### IV.B.3. Design of Data Processes, Processing Logic

The design of Data Processes focuses on the functional aspects and includes the design of procedures for: data ingestion, transformation, production, publication, etc. This activity is the classical Extract Transform Load, ETL, and/or Extract Load Transform, ELT of data warehouses and data lakes.

So that the **Processing Logic activities** can be those of:

- **Data Ingestion, gathering, harvesting, grabbing, web page crawling, etc.**
- **Data Transformation, transcoding, decoding, converting, reformatting,...**

- **Data load/save to storage, retrieve from storage**
  - the load/save activity is typically performed loading (sending) data to an Internal Orion Broker V2, or on some MyKPI storage. In both cases, the data arrives into the OpenSearch.
  - the retrieval is typically performed using one of the several query / search nodes provided from the Snap4City Library on Node-RED.
  - Many other kind of storage connections are accessible in Snap4City Processing Logic (IoT App). For example, with Azure, MySQL, ORACLE, AS400, etc.
- **Data production, generation, etc.**
- **Data publication, post in other channels of any kind, etc.**
- **Server-Side Business Logic as described in the following.**



**Figure 17: a Node-RED with multiple Folders/Labels, and in the current folder Examples you have 5 independent flows.**

In Snap4City, the activity of **Processing Logic** is strongly simplified since all these functional aspects are easily developed via Node-RED flows exploiting a large set of nodes from Snap4City libraries. This approach is based on visual programming where the usage of JavaScript is quite limited. In Node-red, there are several libraries to perform all kinds of functions in the above categories and much more. Snap4City implemented 4 Node-Red libraries dedicated to Smart City IoT platform: <https://flows.nodered.org/search?term=snap4city>. **Processing Logic** is implemented in Node-RED in which you may have different folders, and in each folder, you may have multiple flows, processes. You can also group nodes, link them, create macros, etc. (see **Figure 17**).

For each **Data Process** in **Processing Logic (IoT App)** / Node-RED one should answer at questions and identify:

- What process must be implemented to process the data/entities?
- Which kind of data formats, protocols and channels will have to be used?
- Which data/entity models would be exploited and produced?
- What are the data transformations required (e.g., transcoding)?
- How many independent data flow are needed?
- For each of them: In/out formats and Data Model if any?

- Complexity and possible nature: ingestion, production, transform, load, etc.?
- Which kind of Node-Red block/nodes can be used, are they available?
- How much data will arrive per day, per month, per year; and thus, which is the volume in terms of byte that I am going to collect?

#### How to proceed:

1. **A Processing Logic should be composed by a set of independent Flows for the scalability and for the simplicity, for example:**
  - **Receiving data from the field** (if you connect data reception with processing means that you module for processing/assessing will lock the reception of other messages)
  - **Assessing the data according to the status and producing results**
  - **Providing notifications**
  - **Preparing reports**
  - **Sending reports**
  - **Computing monthly update**
  - **Receiving events from dashboards, Sending reaction to dashboards**
  - **Etc.**
2. Decompose your problem and sequence diagram in a set of **independent flows as much as possible**. An **independent flow** is one that **starts** from a certain event (periodic or provoked from the arrival of a message into a broker (ORION or MQTT), get some data, do some transformation, and finishes with one or more actions such as:
  - **start** of a flow can be based on event, you can realize Event Driven flows. In Snap4City Events can come from:
    - Brokers: MQTT, Orion Brokers, or others...as well as from any listener-based protocol
    - MyKPI changes via Snap4City nodes
    - Dashboard events via Snap4City nodes
    - Manual start via an Inject Node
    - Programmed periodic start from an Inject Node
    - Eventual HTTP call API, but its strongly discourages as state in other paragraphs.
    - Etc.
  - **actions/activities** such as to
    - get some data from storage via search node of Snap4City Library
    - perform a data transformation, etc.
    - activate the Data Analytic process, etc.
    - send a new Entity Message in the ingestion Broker (e.g., NGSI V2 Orion broker) and thus into the storage to change the Entity Instance status,
    - send an event to some broker or MyKPI (this can provoke action on Dashboards elements and on Synoptics),
    - provide a message into a widget dashboard for directly providing some changes into the user interface,
    - send an email, or a telegram,
    - etc.
3. Design the single **independent flows** with a mixt of the possible **activities**.
  - The design can be performed using data flow diagrams, which are stream programming based.
  - It may have sequences, switch, serialization (split), packing (join), distribution, communication, transformation, search, etc.
  - The classic FOR programming construct to select single data structures over a vector of results is typically substituted with Split/Join, and thus processing them on single JSON message/structure. If you need you can have FOR into the Function Node in JavaScript.

4. When the design of **independent flows** mechanism is clear the designers can pass to directly sketch the flow in Node-RED which is a visual programming.
5. **Incrementally** improve the Processing Logic (IoT App) Node-RED flows by adding nodes needed.
6. **Eventual communications among different Processing Logics** (IoT Apps) can be implemented as:
  - **Asynchronous** nonevent driven by using storage.
  - **Even driven** by using MyKPI or Orion Broker

#### IV.B.3.a- Thumb Rules on Snap4City Processing Logic development

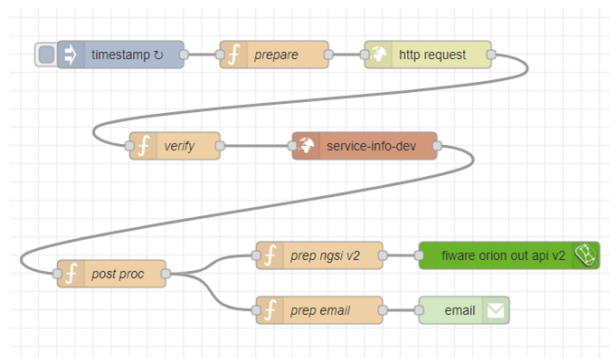
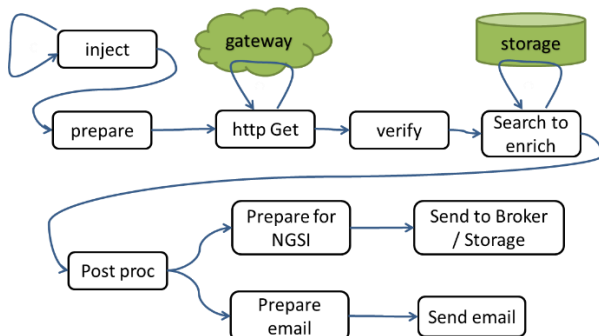
1. Take in mind that using asynchronous stateless rest calls is the best modality to work with MicroServices.
2. Each Rest Call of the Snap4City Environment and thus each Node in Node-RED should be stateless on the dialogue among systems and subsystems.
  - First Node-RED tutorial: <https://nodered.org/docs/tutorials/>
  - Using editor: <https://nodered.org/docs/user-guide/editor/>
  - Training on the main core nodes: <https://nodered.org/docs/user-guide/nodes>
  - Once done we suggest you to pass at the Snap4City Training:
    - <https://www.snap4city.org/download/video/course/iot/>
3. Each Node in node-RED Proc.Logic:
  - Receive messages from the single INPUT on the “msg”.
  - Send a number of messages from its outputs which can be 1 or more according to the Node design. The Function node may provide multiple Output according to the setting performed on visual programming/configuration of the specific node.
  - All the messages msg are JSON formatted, and may contain other data kind in ASCII code, such as HTML, XML, CSV, encoding, etc.
  - The msg (In/OUT) can have substructures such as msg.payload, msg.topic, and the developer can add other custom.
4. You may have temporary data storage into the processing logic at level of context, flow or processing logic global for data exchange among tabs by using constructs as follows. These constructs should be used carefully since the data are not saved if your Processing Logic is restarted as happen periodically on cloud. MyKPI and Entities can be used for safe status changes and long terms storage. So that be careful in using.
  - **context** means into the subflow
    - Var nax, nax2;
    - context.set("name",nax); // to temporary store a variable
    - nax2 = context.get("name"); //to retrieve a variable
  - **flow** means into the Folder Flow
    - Var nax, nax2;
    - flow.set("name",nax); // to temporary store a variable
    - nax2 = flow.get("name"); //to retrieve a variable
  - **global** means into the Folder Flow
    - Var nax, nax2;
    - global.set("name",nax); // to temporary store a variable
    - nax2 = global.get("name"); //to retrieve a variable
5. Each Entity (virtual of physical) may have a status saved on Storage (via Orion Broker node V2), and the status can be used to align subsystems to the evolution of the dialog among entities, allowing to keep trace of the versioning and time evolution, aka time series.
6. improve your Processing Logic (IoT App) which are Node-RED flows by adding nodes needed, taking in mind that you can:
  - test them step by step on deploy,
  - save/load and share the flows in JSON files,
  - collaborate on editing with other colleagues sharing the same account,
  - load in your Processing Logic (IoT App) any kind of examples from public Node-RED library from



- JS foundation, via menu and Import function,
- Install in your Processing Logic (IoT App) any kind of Node-RED library from JS foundation, via menu and Managing Pallet,
- create your own microservice/node for API using the Snap4City tools,
- etc.

#### IV.B.3.b- Example of Processing Logic (IoT App) Design, for each independent Flow

- 1) Identify the activities to be performed, for example:
  - a. Periodically activate the flow.
  - b. Call a gateway to get data.
  - c. Verify the correctness of data.
  - d. Enrich the data with other information coming from Cloud data into the storage.
  - e. Transform the data in the correct format.
  - f. Send the data into the Broker, and thus send the data in the storage on a specific entity Instance / IoT Device.
  - g. Send also a notification via email.
- 2) Sketch a data flow diagram (see **Figure 17b left**).
- 3) Implement the sketch on processing low using Node-RED nodes, Snap4City lib of nodes and other libraries of nodes. (see **Figure 17b right**).
- 4) Deploy and test using DEBUG node in any stream you like, using also on/off to avoid deploy all time.



**Figure 17b – Single Processing Logic Independent flow: (left) design, (right) actual implementation in Snap4City of the flow in Node-Red tool.**

#### IV.B.3.c- Main Snap4City Proc.Logic / IoT App nodes/block for Node-RED

Please note that, the most important blocks nodes to interact with the platforms are reported in the following table to familiarize with the main concepts. **They are actually families of blocks/nodes** since many other similar nodes are present that allow you to perform a large number of other features. The main principle is that YOU SHOULD NOT NEED to use HTTP REST CALL to API, since all of them are provided in terms of NODEs/BLOCKS (aka microservices) into the Processing Logic environment which is based on Node-RED plus Snap4City Libraries (IoT APP).

Most of the Nodes / MicroServices are controlled by sending them in input JSON messages with parameters. Most of the nodes can be also configured once from their graphic user interface settings but the JSON is the primary mode for setting parameters. This means that if a node receive data in its input port in JSON, the information and setting on graphic user interface is ignored.

| Node shape | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Snap4City or standard |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
|            | To <b>generate injection messages</b> into a flow, scheduled/periodic or on manual demand by click it on left.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | standard              |
|            | <b>DATA TRANSFORM</b><br>A <b>JavaScript function</b> , from a JSON input to one or more JSON outputs, which can be produced by setting it.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | standard              |
|            | <b>SAVE to STORAGE and BROKER</b><br>To send an Entity Message of an Entity Instance into the storage. The Entity Instance has to be registered on Entity Directory (IoT Directory) and you have to be the owner or to be delegated in READ-WRITE to send messages to it. The node represents the broker, so that the same node can be used to send any Entity Message you need.                                                                                                                                                                                                                                                                                                                                             | Snap4city             |
|            | <b>SUBSCRIBE</b> to an Entity change on BROKER<br>To subscribe the Processing Logic (IoT App) to receive event-driven notifications related to Entity Instances changes. The node is substantially a listener connected to an Orion Broker. You can subscribe to many Entities and then to get all of them from the output of the listener. The new version will go to provide an input port to send at this listener multiple subscriptions.<br><b>PLEASE NOTE THAT ALL THAT YOU CAN DO IN MQTT CAN BE DONE IN ORION BROKER NGSI. Moreover, Orion broker is authenticated, in SSO, provides JSON, etc. This node-red block allow you to subscribe to a topic / device and get event driven actions on IoT App directly.</b> | Snap4city             |
|            | <b>READ from STORAGE</b><br>Query call to Smart City API to get any information about a SURJ, ServiceURI. There are many other Nodes which can be used to pose Smart City API queries in very simple manner and recover vectors of ServiceURIs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Snap4city             |
|            | <b>SEARCH on STORAGE</b><br>To perform queries on the storage to obtain a list of ServiceURI. The nodes of this family can allow you to perform searching queries by filtering for distance, area, subnature/category, values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Snap4city             |

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |           |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
|  | of attributes, time period, etc.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |
|  | Send email. With other nodes you can send Telegram, SMS, etc.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | standard  |
|  | To send a <b>REST CALL</b> (get, post, etc.). Please USE THIS NODE ONLY for the access at external API and not to access at the Snap4City API for which a lot of MicroServices are accessible as NODEs/Blocks in the Processing Logic and they are simpler to be used and ready to use.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | standard  |
|  | A block which is printing on <b>debug</b> view the data JSON passed in its input. Please note that the node can be tuned to provide only msg.payload or the full JSON message, change configuration of the node.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | standard  |
|  | To <b>create an Entity Instance</b> (device instance) from a model prepared on Entity Directory (IoT Directory).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Snap4city |
|  | To <b>change the ownership</b> of an Entity Instance (IoT Device).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Snap4city |
|  | <p>To <b>delegate a certain Entity Instance</b> (IoT Device) to some other user for which you have to know the Nickname. Delegations can be: Read_access, Read_write, Modify (to modify the Entity Instance structure).</p> <p>Please use <a href="https://flows.nodered.org/node/node-red-contrib-snap4city-user version 9.47">https://flows.nodered.org/node/node-red-contrib-snap4city-user version 9.47</a> or newer!.</p> <p>Delegation can be performed to a <b>single user</b> by knowing the nickname of the user. DO NOT put there your user name, Put the nickname of the user at which you would like to grant the access. "username": "username",</p> <p>Delegate to <b>groups</b> of users by specifying the target group according to the LDAP notation at "groupdelegated" such as:<br/>cn=&lt;GR&gt;,ou=&lt;ORG&gt;<br/>where &lt;ORG&gt; is your organization, and &lt;GR&gt; is the group name.</p> | Snap4city |
|  | To <b>show something on Snap4City dashboard</b> with a single content widget (one of the simplest widgets). A large set of dashboard nodes/widgets to send and retrieve data to/from dashboards are provided. This specific Nodes allows to send on dashboard HTML formatted messages with some limitations. Full HTTP widget is also accessible.<br>See in the following section for the Full list of Nodes for Snap4City Dashboard Widgets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Snap4city |
|  | <p><b>MQTT broker listener</b>, to receive messages from the Broker. Another similar node can be used to send MQTT messages to the MQTT broker. This node allows to perform a subscription to a topic of the MQTT broker.</p> <p><b>PLEASE NOTE THAT ALL THAT YOU CAN DO IN MQTT CAN BE</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | standard  |

|  |                                                                                                                                                                                                                                                                |           |
|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
|  | <b>DONE IN ORION BROKER NGSI. Moreover, Orion broker is authenticated, in SSO, provides JSON, etc.</b>                                                                                                                                                         |           |
|  | <b>DATA ANALYTICS</b><br>Request performed on a Container including a Python data analytics, which is loaded into the node and the container is created at the first Deploy of the Processing Logic. Similar Approach is performed for RStudio Data Analytics. | Snap4city |
|  | <b>SPLIT:</b> This block takes in input a buffer, or an array, or an object and split it on a set of messages in output, for each line in the buffer, each element of the array, each element in the object, respectively.                                     | standard  |
|  | <b>JOIN:</b> This block takes in input a set of messages and join/merge them into a single message (string, buffer, array or object, etc.), on the basis of specific criteria.                                                                                 | standard  |

#### IV.B.3.d- The power of Snap4City Libraries for Proc.Logic / IoT App Node-RED

To satisfy the smart city requirements, in Snap4City, a collection of more than 190 MicroServices, as Nodes for the Node-RED programming environment, has been developed.

- <https://flows.nodered.org/search?term=snap4city>
- <https://flows.nodered.org/node/node-red-contrib-snap4city-user>
- <https://flows.nodered.org/node/node-red-contrib-snap4city-developer>
- <https://flows.nodered.org/node/node-red-contrib-snap4city-d3-dashboard-widgets>
- <https://flows.nodered.org/node/node-red-contrib-snap4city-tunnel>
- <https://flows.nodered.org/node/node-red-contrib-snap4city-milestone>

The Node-RED philosophy of visual programming allows for the creation of event-driven data flow applications, where the exchanged messages are in JSON format. On the other hand, periodic processes can also be developed by scheduling one or more internal timers. This means that users can develop Proc.Logic / IoT Apps as Node-RED flows, exploiting both Push and Pull data protocols, in the same visual programming environment. In the context of smart cities, both protocols are needed, while Proc.Logic / IoT Apps are capable of creating flows and exploiting a large number of features that are typically not available in the Node-RED open library, nor in several libraries from different providers. Moreover, the Snap4City MicroServices are at a level that can allow even non-expert users to easily develop Logic / IoT Apps for smart cities.

Actually, there are more than 180 nodes/blocks in the Snap4City libraries on IoT App which can really facilitate your life and save your time in producing Smart Applications for composition of the following microservices and using those that you can install from internet, thousands of functionalities:

- **Data ingestion:** more than 130 protocols IoT and Industry 4.0, web Scraping, external services, any protocol database, etc.
- **Data access:** save/retrieve data, query search on expert system, georeverse solution, search on expert system Km4City ontology, call to Smart City API, etc.
- **Data Transformation/transcoding:** binary, hexadecimal, XML, JSON, String, any format
- **Integration:** CKAN, Web Scraping, FTP, Copernicus satellite, Twitter Vigilance, Workflow OpenMaint, Digital Twin BIM Server, any external service REST Call, etc.
- **Manipulation of complex data:** heatmaps, scenarios, typical time trend, multi series, calendar, maps, etc.

- **Access to Smart City Entities and exploitation of Smart City Services:** transport, parking, POI, KPI, personal data, scenarios, etc.
- **Data Analytic:** managing Python native, calling and scheduling Python/Rstudio containers as snap4city microservices (predictions, anomaly detection, statistics, etc.)
- **User interaction on Dashboard:** get data and message from the user interface, providing messages to the user (form, buttons, switches, animations, selector, maps, etc.), send data to special graphical widgets: D3, Highcharts, etc.
- **Custom Widgets:** SVG, synoptics, animations, dynamic pins on maps, etc.
- **Event management:** Telegram, Twitter, Facebook, SMS, WhatsApp, CAP, etc.
- **Special tools as:** routing, georeverse, verify if a point is included into an area, Twitter Vigilance and sentiment analysis, get the closest civic number, distance from two GPS points, etc.
- **Hardware Specific Devices:** Raspberry Pi, Android, Philips, video wall management, etc.
- **Etc. etc.**

Moreover, the nodes in Snap4City can be also created by mapping some external API into a node. Thus, for the automated production of Nodes for Node-RED from API see:

- Creating new microservice nodes: <https://www.snap4city.org/217>
- tutorials of Node-RED. <https://nodered.org/docs/creating-nodes/>
- creating a new microservice node in node-red from some Rest Call API:
  - <https://www.snap4city.org/129>

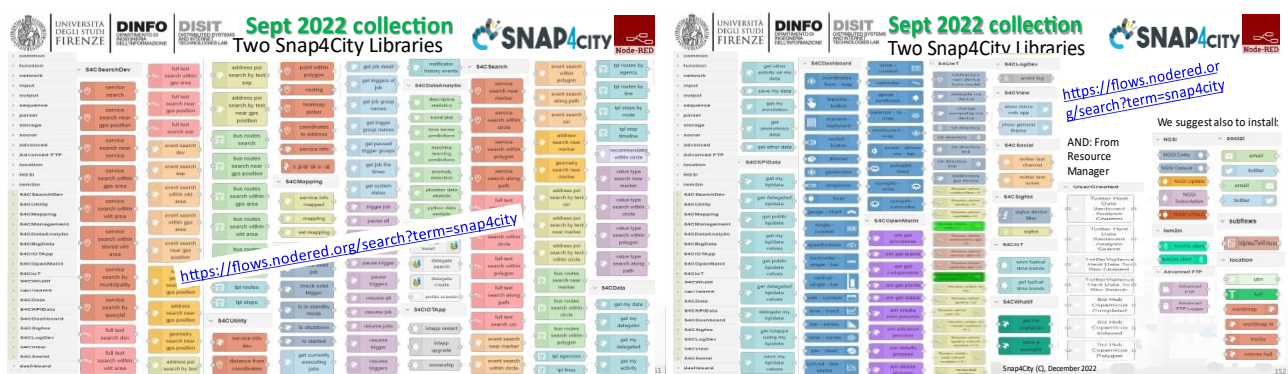


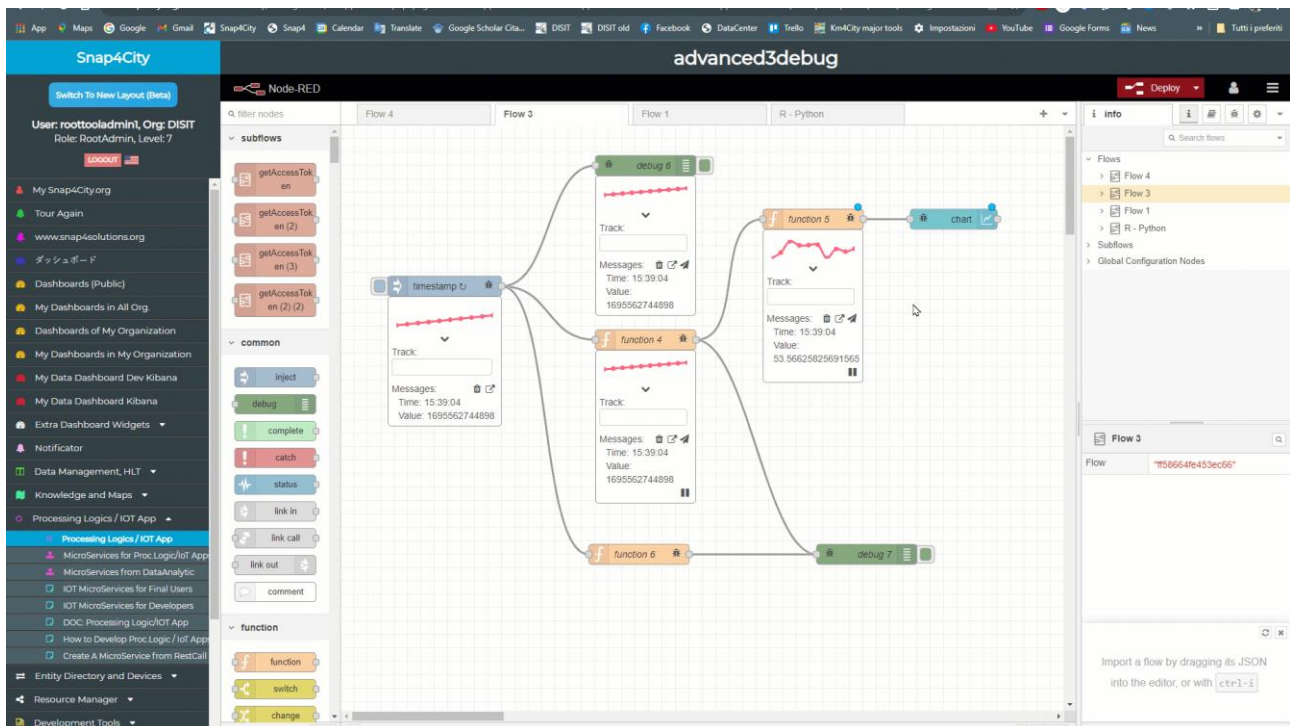
Figure 18 – Overview of the Snap4City MicroService / node for Node-RED

Please note that, the Entity Instances (IoT Devices) produced by a user can be delegated in read or read-write to other users. All the Entity Instances start as private of the user who created. This means that, the administrator can create the Entity Instances of the user profile and then delegate the access, or fully pass to him the Entity Instance ownership, losing the possibility of accessing to the Entity Instance.

#### IV.B.3.e- Snap4City Proc.Logic / IoT App with Debug Option

Snap4City added the extended debug capabilities, by using and extending CAULDRON tool. So that it is possible to attach at each Node extended debug capabilities. The integration of the CAULDRON offers the possibility of monitoring the passage of messages entering the Node-RED nodes, also providing the possibility of using breakpoints to block the flow through a node in order to be able to analyze it, make it proceed step-by- step, release the message queue held by the breakpoint or delete it. Specifically, the nodes of the CAULDRON version using the debug icon it is possible to see the trend of the last 10 messages. The graph draws lines if the plotted field contains an integer to highlight the value trend, otherwise it shows points to highlight the temporal trend. Clicking on the down arrow further modifies the interface. Nodes with both input and outputs provide other functionalities.





<https://www.snap4city.org/964>

The three icons next to the messages label have the following functions:

- basket icon allows you to delete the entire history of messages received and sent by the node;
- link icon to open a pop-up that shows the history of messages received and sent by the node;
- airplane icon to inject a personalized message into the node.
- pause icon puts the node at breakpoint, allowing you to pause the flow. Messages arriving in the meantime are placed in a queue.
- Fast Forward icon allows you to extract the first message from the queue and let it continue the flow.

The Value field is an editable textbox containing the Json of the first message in the debug queue. Snap4City enhanced version with respect to the original version of Cauldron allows you to select, through a special textbox present in each node, the attribute to be tracked. The permitted syntax is the usual JavaScript one, therefore for an object of the type: `{A:[{B:'0b', },{B:'1b', }]}`

Typing the expression `A[0].B` into the textbox shows the value `'0b'` while the expression `A[1].B` shows the value `'1b'`. If the expression does not match any attribute of the object, no results are shown. This functionality can be used in static ('static') or dynamic ('dynamic') mode: static mode means that the change of the textbox takes effect from the next message that crosses the node, while dynamic mode means that updating the textbox involves an instant update of the value displayed on the screen. If the textbox remains empty, the value displayed will be the default one, usually the message arrival timestamp for generic nodes.

The features just discussed are accessible via a dropdown UI specific to each node, which can be activated via the respective debug button on the node.

For the purposes of extending the graph, please note that it stores and takes into account only the last 10 messages received, and NOT the complete history since the activation of the flow.

To USE Snap4City Proc.Logic with Debug Option based on extended Cauldron please ask to <mailto:Snap4City@disit.org>

We suggest to use those debug versions only for debug and develop while for execution use the classic Basic and Advanced versions provided by Snap4City, since they provide higher performance in execution.

Very soon, nap4City Proc.Logic with Debug Option will be provided to all users on Snap4City.org ONLY.

#### IV.B.3.f- Why and How to use delegations

Each entity in Snap4City when is created is only visible and can be edit only for its owner, the so-called Creator. The Owner/creator can pass the ownership to one or more users of the same Organization, or to one or more Groups of the Organization. The Groups in the organization can be created by the Root Administrator of the platform, please ask.

The screenshot shows the Snap4City web interface. The left sidebar contains navigation links such as 'My Snap4City.org', 'Tour Again', 'www.snap4solutions.org', 'Dashboards (Public)', 'Dashboards of My Organization', 'My Dashboards in My Organization', 'My Data Dashboard Dev Kibana', 'Extra Dashboard Widgets', 'Data Management, HLT', 'Knowledge and Maps', 'Processing Logics / IOT App', 'Entity Directory and Devices', 'Resource Manager', 'Development Tools', 'Management', and 'Decision Support Systems'. The main content area displays a welcome message for a new user, 'paolo.disit', and suggests various actions like 'We suggest you:', 'Congratulations!', and 'You have reached a level in which you can contribute with competence to the city improvement and smartness'. The right sidebar includes a search bar, a 'Training on Tools and Platform' section, and a list of 'Organization Groups' including 'DISIT Developer' and 'Operative'.

Once the Group has been created, the users of the Organization can join the group by means of the user interface entering in the group, the list of activate groups appear on the right side of the web page.

The screenshot shows the Snap4City web interface in the 'Developer' section. The left sidebar is identical to the previous screenshot. The main content area displays the 'Developer' section with a 'Group' dropdown menu and a 'Subscribe to group' button. The right sidebar is also identical to the previous screenshot.

To this end, each entity in Snap4City provides a specific graphic user interface to change the ownership for: IoT Device Models (Entity Models), Entity Instances (IoT Devices), Dashboards, IoT Apps, etc.

Once the Ownerships is passed, the former Owner/creator lost the possibility of editing and view the entity. Each entity has its own editor for the creation. For example, Dashboards are created by the Dashboard manager, IoT Devices (Entity Instances) from the IoT/Entity Directory. So that the Owner of an entity is the only person which can delegate of the user/groups to access/read or write to a Device/Entity, and there is a specific Manager button or button in the graphic user interface to do it. Moreover, for devices/instances, it is also possible to script the delegation on IoT App/Proc.Logic as follows.

The Owner of an IoT Device/Entity Instance can provide some grant rights to other users.

- **READ\_ACCESSSS:** means to be capable to read data messages of a device/Entity Instance even if you are

not the Owner. This grant allows you to create devices/entities which can read from one or several users.

- **READ\_WRITE:** means to be capable to send new messages on that device/Entity, and also to read the data provided. This grant allows you to create devices/entities which can receive messages / data from one or several users.
- **MODIFY:** means to have the right to modify the device/entity structure. This grant is quite strong and should be carefully used and in general the Owner should be conceptually the only one authorized to change the device / entity structure.

#### Patterns:

- A user A creates a device, and post data for it, and it is interested to communicate the data to many users at which the user A provide READ\_ACCESS.
  - To this end, A has to know the Username of the platform to create the delegations.
- A user A creates a device to receive messages notification from many users. A sort of mailbox for receiving some event notifications.
  - To this end, user A provide READ\_WRITE grant to each of them. They are going to write their messages on the same mailbox, with the hope to avoid them to send messages at the same time stamp.
  - User A can read the message notification and can overwrite them to confirm their reception.
- If all the IoT Devices/Entity Instances produce by a given Device Model are for instance userprofiles of some application, they can be searched and listed by all users having at least the READ\_ACCESS to those devices.
  - The platform provides a Search block in IoT App / Proc.Logic, as well as Smart City API as query by model.
  - In both cases, the users performing the query will receive back only the devices/entities he/she created and those that have been delegated to him.

The recipient of a delegation (the target, the delegated) can be a single user via its username/nickname or the group name by using its LDAP definition, see previous tale for an example.

#### IV.B.3.g- Blockchain support on Snap4City

The Snap4City blockchain can be used in multiple domains. For example, in the context of mobility and transport it could be used to certify the way we interact with the world, connecting everyday devices/entities, in the cities and industries to certify: (i) distribution of goods and services in last mile collaborative framework, such as city hubs; (ii) vehicle identity, mileages, maintenance operations, reparation from incidents, on board unit data for insurances, on board unit data for professional drivers (to certify their readiness and attention level); (iii) tickets sold on the Mobility as a Service platforms, MaaS, also using it for revenue sharing towards the operators of multimodal traveling; (iv) computation of KPI indicators of the city with the aim of comparing them with other cities and for taxation, etc. In environmental applications, it can be used to certify measured values of pollution metrics, which are used for city taxations and to assess the achievement of the targeted values according to the European Commission. In the context of health to certify the source of organs and blood and their delivery chain of cold, the process for cleaning surgery instruments and hospital sheets, etc. In the context of energy and gas, it can be useful to certify the energy produced and exchanged in the communities of energies. It can also be used to certify, unicity, ownership and/or provenance of digital content, art, collectibles, and more through NFTs (Non-Fungible Tokens) that are unique digital assets authenticated on an immutable ledger. Blockchain can enhance smart home applications in regard to the fundamental security goals of confidentiality, integrity, and availability. In the literature many specific frameworks have been developed, while the IoT/WoT platforms may be used to implement all those cases. IoT/WoT platforms are capable to manage device messages exchanged with any kinds of data/device structure. In this context, a strong push on defining data models has been realized, for

example the FIWARE Smart Data Models, SDM. A data model provides a formal template format for IoT/WoT entity/messages with formalized variables/attributes including data types, units, etc., to produce identical entities, and this process should be certified as well. As to the platforms under analysis, messages from IoT devices are freely shaped, to assure data flexibility. For example, IBM Watson uses formats such as JSON or XML, without supporting FIWARE SDMs, which also are published in several versions over time. The IoT/WoT models are the formal template from which the devices/entities can be generated/registered, and in turn the devices/entities with their registration formalize the structure of the messages which can be received/sent on them by the brokers and storage. Whenever a message arrives from a device (which can partially provide pieces of information into its body, typically not the metadata, since most devices minimize data transmission), the platform is not capable to register the device, nor to correct the message link to former devices.

Snap4City added blockchain on IoT/WoT infrastructure adding a set of general-purpose features for certification/verification, and at the same time leaving flexibility to the developers of IoT/WoT solutions, allowing them to use the blockchain technology to set up their scenarios with a mix of certified and non-certified entities in a federated distributed architecture. The main contributions are on: (i) certification of any kind of IoT/WoT data models, devices and messages (with a particular focus on the certification of data models and its implications); (ii) automation of certifications according to the relationship among models and devices/entities, and among devices/entities and messages to guarantee consistency; (iii) certification of the hash for data messages by defining specific rules to cope for the flexibility of JSON objects of the simple schema validation; (iv) certification of time series, within a certain interval, for example to certify a trip, a mission, a delivery travel, etc.; (v) an architectural multi-organization solution to guarantee satisfactory performance in the certification of messages which is the most critical performance aspects.

The blockchain support is accessible via Entity/IoT Directory in which the developer can:

- Specify which Entity Models have to be certified, implying the production of Entities / Devices which are certified.
- Specify which Entities Instances / IoT Devices have to be certified, implying that messages on them will be certified
- Verify the certifications performed over time, on the time series of the devices / entities.

The developers can use these facilities to implement a large range of applications: voting, access rights, rewards, revenue distribution, healthiness, driver assessment, emission of coupons, digital collection, NFT, complimentary currencies, loyalty tools, crowdfunding, etc.

Ask to [Snap4City@disit.org](mailto:Snap4City@disit.org) to get information and support to use this advanced feature on Snap4City.org infrastructure and on MicroX installations.

See: <https://www.snap4city.org/965>

#### *IV.B.3.h- Snap4City integrated with Milestone X protect VMS*

The actual integration of Snap4City with Milestone is performed by means of a node-red library of Snap4City:

To satisfy the smart city requirements, in Snap4City, a collection of more than 190 MicroServices, as Nodes for the Node-RED programming environment, has been developed.

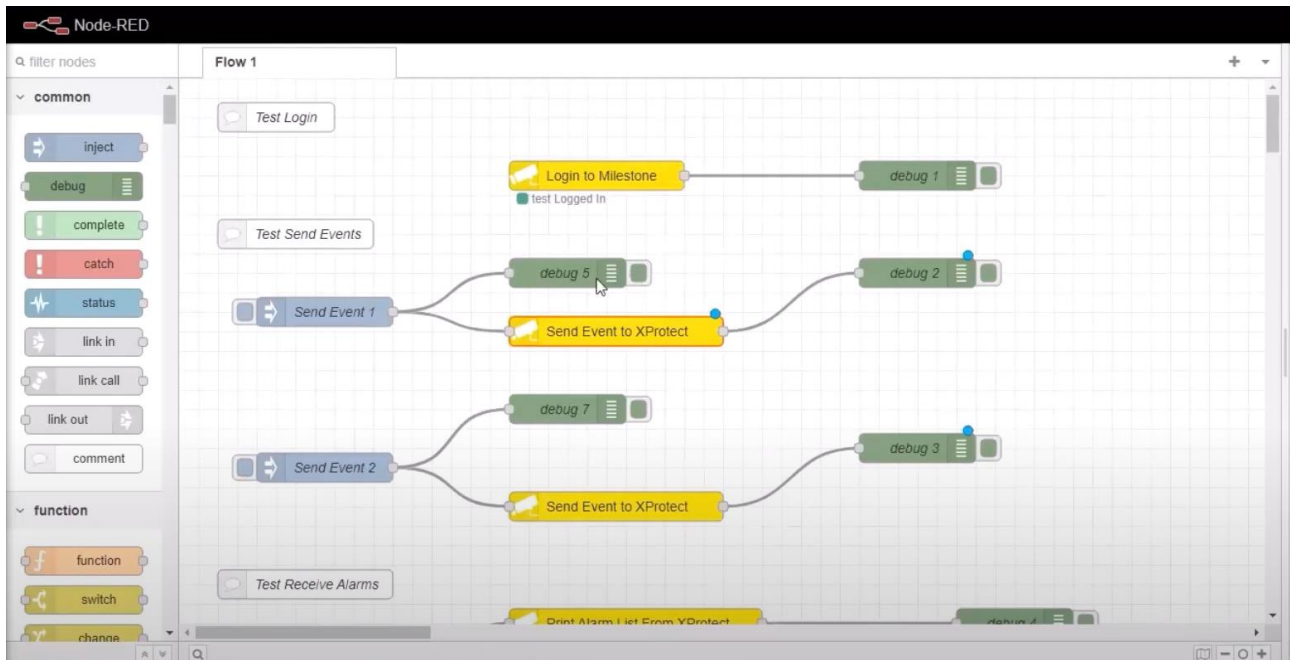
- <https://flows.nodered.org/node/node-red-contrib-snap4city-milestone>
- See video: <https://www.youtube.com/watch?v=dxxYtOKOkvE&t=7s>

The Snap4City node-red library can be used even without the usage of Snap4City tools. It allows to:



- send events from Node-RED to VMS X protect of Milestones
- receive events from VMS X protect of Milestones into Node-RED

The library can be installed on AXIS camera, on windows, Linux, raspberry Pi, etc., without the installation of .net.



The Snap4City platform can be integrated with Milestone VMS to create a more complete and integrated surveillance and security system. Here are some examples of how the Snap4City platform could be useful in combination with Milestone VMS:

- **Traffic monitoring and management:** The Snap4City platform can be used to integrate traffic information from Milestone VMS security cameras and display it on an interactive map. This allows you to monitor traffic in real time and manage it more effectively.
- **Incident Management:** In the event of road accidents, the Snap4City platform can be used to integrate information from Milestone VMS security cameras with other data sources, such as traffic data and weather conditions. This allows you to coordinate emergency resources more quickly and improve incident management.
- **Public Safety:** The Snap4City platform can be used to integrate information from Milestone VMS security cameras with other data sources, such as weather data and social media data. This allows you to monitor public events in real time and improve public safety.
- **Parking Management:** The Snap4City platform can be used to integrate information from Milestone VMS security cameras with other data sources, such as traffic data and parking data. This allows parking to be managed more efficiently and improves the availability of parking for citizens.

In summary, the Snap4City platform can be used in combination with Milestone VMS to create a more complete and integrated surveillance and security system. The platform allows information from different sources to be integrated and displayed on an interactive map, improving traffic management, public safety and parking management, among other things.

Node-RED is a visual programming platform used for creating IoT (Internet of Things) applications based on data flows. Node-RED can be used to integrate Milestone VMS and Snap4City, providing a way to process,



analyze and visualize Milestone VMS camera data within Snap4City. Here's how I see the use of Node-RED between VMS and Snap4City:

- **Data Processing:** Node-RED can be used to process Milestone VMS camera data and transform it into a format compatible with Snap4City. For example, Node-RED can be used to analyze motion data from Milestone VMS cameras and send only relevant information to Snap4City.
- **Data Analysis:** Node-RED can be used to analyze Milestone VMS camera data along with other Snap4City data sources, such as traffic data and weather conditions. This allows you to extract more meaningful information and provide a more complete view of the situation.
- **Data Visualization:** Node-RED can be used to send processed and analyzed data to Snap4City for display on an interactive map. This allows you to monitor traffic and safety in real time and make more informed decision

Furthermore, Node-RED offers a wide range of libraries and tools to easily integrate different data sources and communicate with different platforms. This simplifies the integration between Milestone VMS and Snap4City, reducing development time and improving overall system efficiency. In summary, using Node-RED between VMS and Snap4City can be very useful for processing, analyzing and visualizing Milestone VMS camera data within Snap4City. This allows you to integrate different data sources and provide a more complete view of the situation, improving traffic management, public safety and parking management, among other things.

#### IV.B.4. Design of Data Analytics

The design and development of Data Analytics, DA, is mainly performed taking in mind that the development can be performed in Python or Rstudio. And they are automatically converted in containers exposing some API/microservices which can be called from Processing Logic (IoT App) node-RED in multiple instances or as singleton. The design of DA implies to decide their aims, for example, for implementing specific algorithms, or making predictions, anomaly detection, suggestions, statical analysis, clustering, classification, detection, recognition, etc. Most of these aims can use techniques from ML, AI, XAI, NLP, operating research, statistics, etc., and exploit a set of libraries from Python or RStudio to produce a DA Model, DAM. For example, from a training phase. Python and RStudio platforms may exploit any kind of libraries such as Keras, Pandas, and hardware accelerator as NVIDIA to use Tensor Flow, etc. The developers can access the KB and Big Data store respecting the privacy and the data licensing by using authenticated Smart City APIs. The access has to permit the reading of historical and real-time data, and saving the resulting data provided by the algorithms, for example, heatmap-related predictions, the assessment of data quality, and labels of detected anomalies.

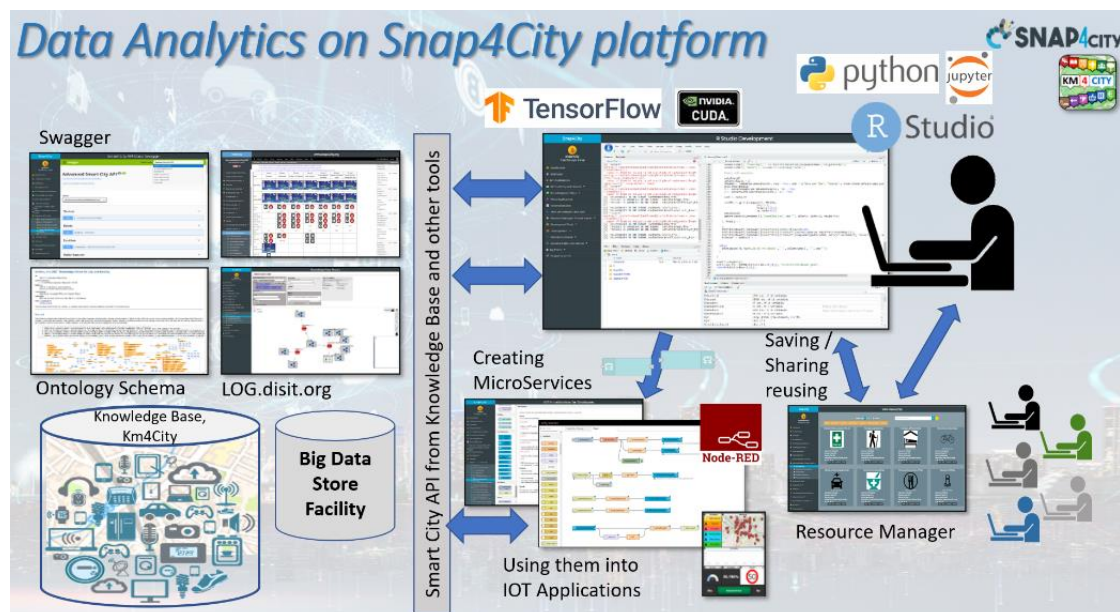


Figure 18b – Schema of Data Analytics development

In Snap4City, this activity is strongly simplified since the Data Analytic can be implemented in Rstudio or Python exploiting processes that are executed on:

- **Dockers Containers** accessing and controlling them via some API, and these can be automatically produced and manage by the platform. The management is typically performed by some Proc.Logic (IoT App).
- **Dedicated servers (for example for developers, or for exploiting high performance boards as the NVIDIA, in the lack of VGPU on your cloud resources)** and leaving them to access to the storage for using the data and providing results via Snap4City API, in authenticated and authorized manner.

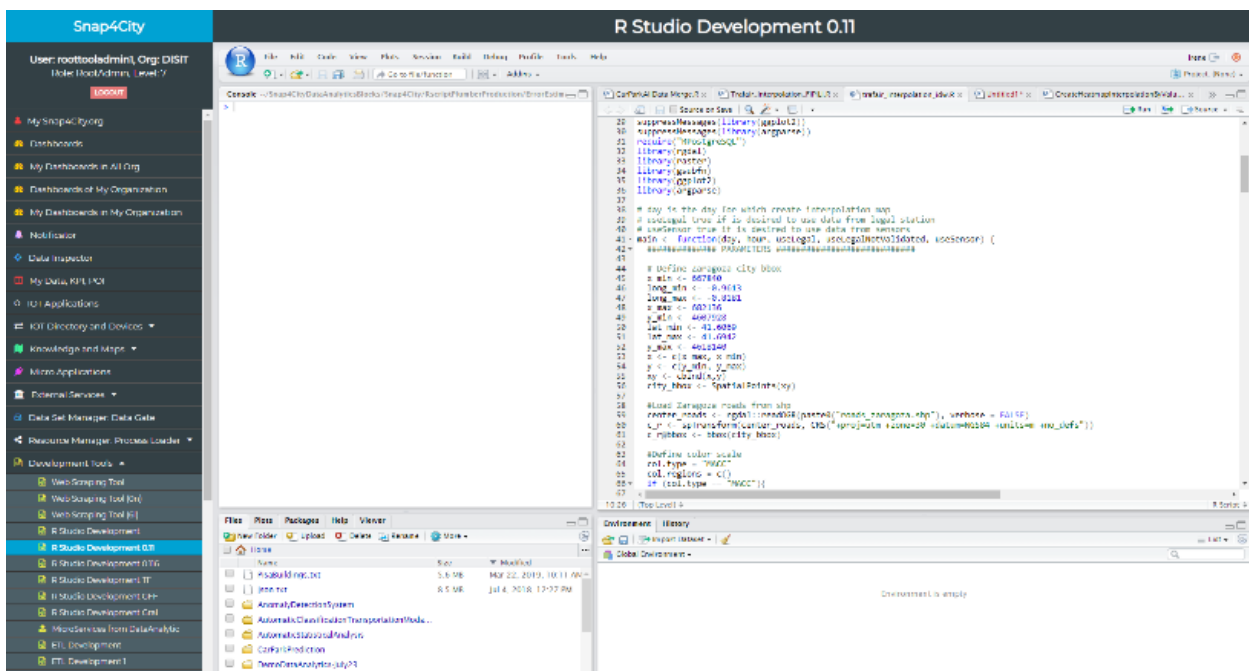
For each **Data Analytics** one should identify:

- What process must be implemented by the Data Analytics?
- Which data models would be produced?
- Which data are needed?
- For each of them: In/out formats and Data Model if any?
- The process to be implemented is for training or for production?
- How many processes for production I am going to have at the same time?

- Which the execution time?
- Which is the expected precision, and which is the state of the art?
- Do I need to execute the Data analytic exploiting special hardware as NVIDIA since I am going to use CUDA, tensor flow, ...?

**How to proceed to design the single Data Analytic according to its nature.**

- Problem analysis, business requirements.
- Data Discovery, Data ingestion, acquisition (as above presented that can give for granted).
- Data set preparation, transformation, identification of features, normalization, scaling, imputation, feature engineering, etc.
- Target Assessment Model Definition
  - Identification of metrics for the assessment, KPI.
  - Typically: R2, MAE, MAPE, RMSE, MSE, MASE, MAP, ...
- Screening on Models/Techniques, for each Model/Technique or for the selection Model/Technique perform the
  - Model/Technique Development/testing
- Best Model selection among those tested
  - If needed reiterate for different parameters, features, etc.
  - Comparison with state-of-the-art results.
  - Needs of Explainable AI solutions: global and local.
- Deploy best Model in production, monitoring in production.



**Figure 18c – Data Analytics development in R-Studio, similar to Python which is in Jupiter HUB**

In Python and/or RStudio cases, the script code has to include a library for creating a REST Call, namely: Plumber for RStudio and Flask for Python. In this manner, each process presents a specific API, which is accessible from an IoT Application as a MicroService, that is, a node of the above-mentioned Node-RED visual programming tool for data flow. Data scientists can develop and debug/test the data analytic processes on the Snap4City cloud environment since it is the best way to access at the Smart City API with the needed permissions. The source code can be shared among developers with the tool “Resource Manager”, which also allows the developers to perform queries and retrieve source code made available by other developers.

Rstudio and Python data analytics processes may include conceptually any kind of libraries for ML, AI, operative research, simulation, etc. The Development environment may be configured to allow at the single operators to load their own preferred libraries. On the other hand, when the process is adopted to produce a contains as in the next figure, the container has to include the library used in the code. This can be performed by requesting a specific image to the platform manager and indicating the library you would like to have on Container executions.

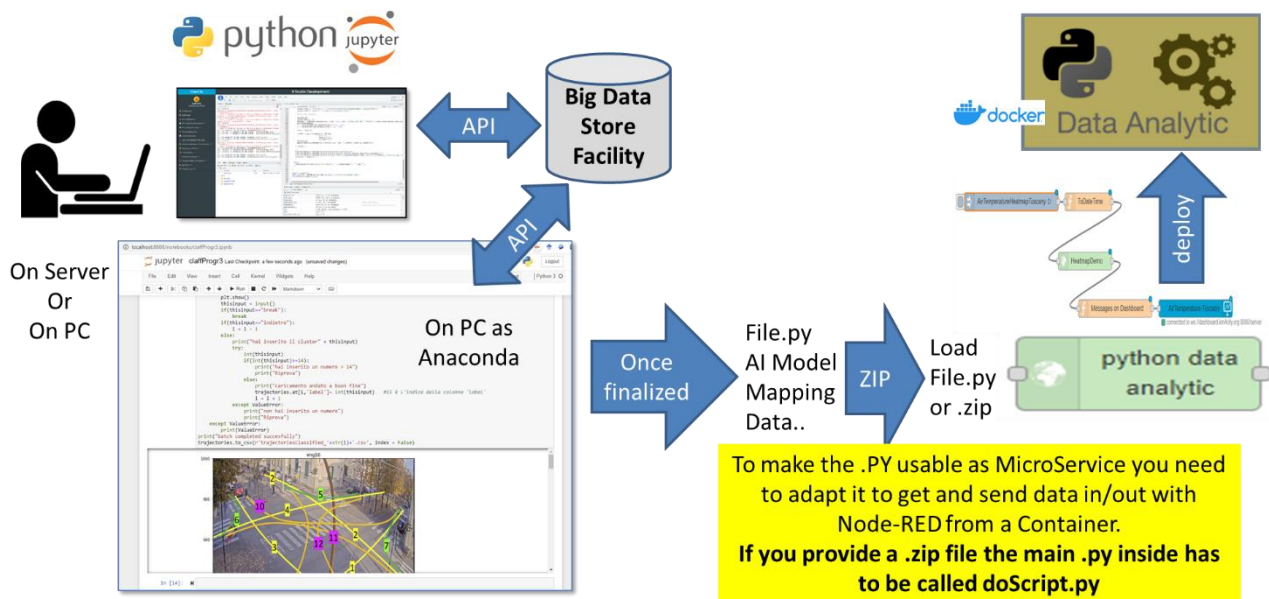
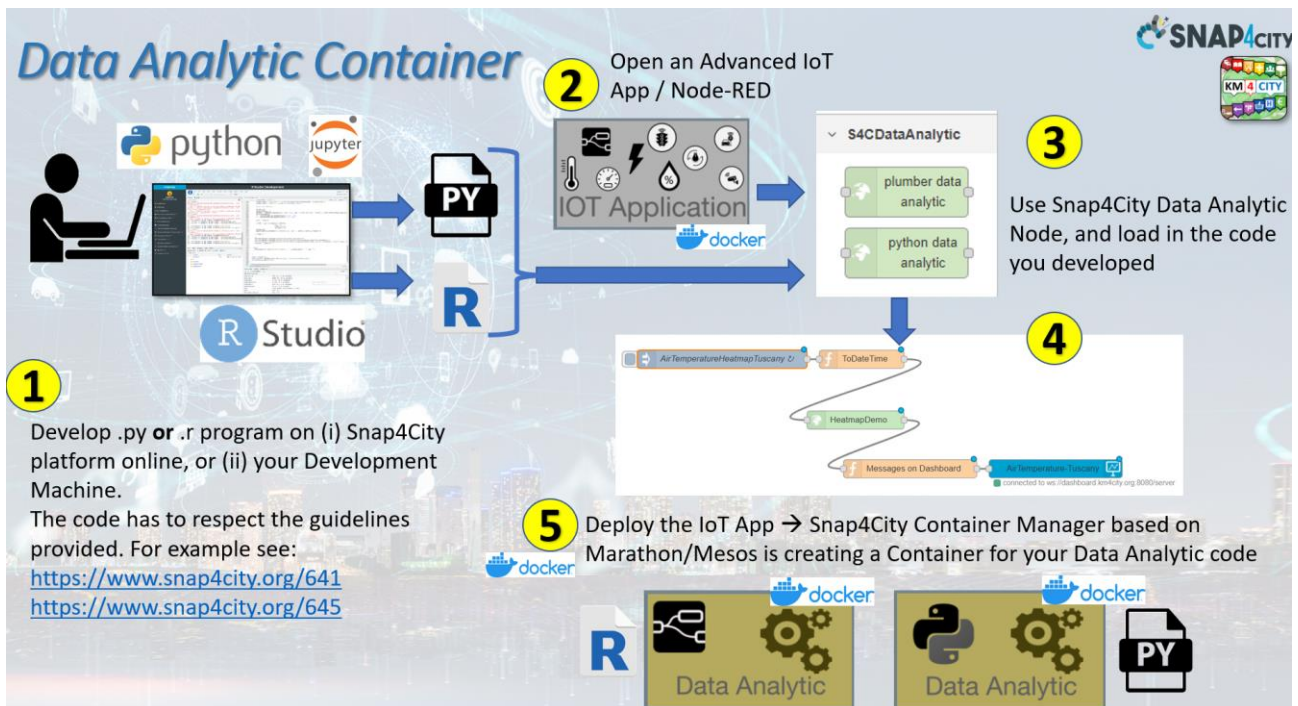


Figure 18d – Data Analytics development flow in Python.

This description of the flow refers to case in which the Python or Rstudio are created to be used as MicroServices as contained from the Proc.Logic / IoT App. An alternative is to develop the Data Analytic to be used as standalone services, working on API, or providing some REST Call, and thus usable from Proc.Logic / IoT App according to the API or by collecting results on database. These aspects are described into the training course.



**Figure 18e – Data Analytics development flow in Python and integration into Proc.Logic / IoT App.**

In Snap4City, there is a **specific tutorial for the Data Analytic** development with several examples:  
<https://www.snap4city.org/download/video/course/da/>

Read the mentioned slide course and/or platform overview to get a list of Data Analytics in place:  
<https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>

We also suggest reading the Snap4City booklet on Data Analytic solutions.  
[https://www.snap4city.org/download/video/DPL\\_SNAP4SOLU.pdf](https://www.snap4city.org/download/video/DPL_SNAP4SOLU.pdf)



### IV.B.5. Design: User Interface and Business Logic

The design of the **User Interface** implies the design of dashboards to be developed. Snap4City Dashboards are composed by several graphical widgets accessing to: Storage, Processing Logic (IoT App) data/nodes, External Services, Synoptics, and Brokers.

**How to proceed:** This phase is performed by identifying **Data Representation and Graphic User Interaction**: This phase has to answer at questions such as:

- Which kind of visual rendering is more adequate? Which kind of user interface I have to provide to the users? Which kind of graphic user interface your users would prefer?
  - Which kind of widget? The answer is easy since the preferred rendering tool for each Data Model has been defined.
- How many users are going to use it?
- It is a scenario for Control Room or for understanding (such as a Business Intelligence tool to play with data and study)?
- Which Entity Instances have to be shown?
- How many dashboards or view I need to create, how large they are, on which device they are shown?
- The user interface is only a Monitoring data from Storage?
- The user interface has to provide data table for browsing on data? And in which order?
- Do we need a menu to navigate on a number of connected Dashboards?
- Who is going to access to those dashboards?

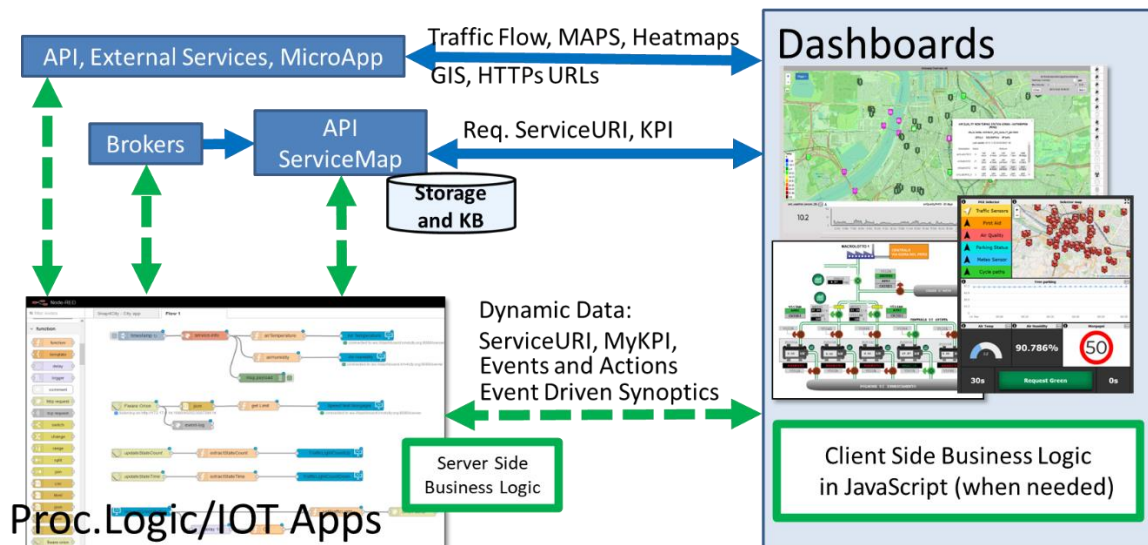
#### IV.B.5.a- *Passive and Active Dashboards/views*

The Dashboards are composed by widgets. Each widget may represent several data and has a specific graphic representation and user interaction. Before stating the design of the user interface, you have to know the capabilities of the Snap4City Dashboards which are very wide providing almost any kind of widgets and graphic representation for your data, and relationships among them to create not only good representations but also a good interaction design, to specific what is going to happen interacting with the graphic elements and data on your user interface.

In Snap4City, there is a specific tutorial for the Dashboard development with several examples and the full list of capabilities in SLIDES: <https://www.snap4city.org/download/video/course/das/>

#### **Dashboard Widgets:**

- are the main components of the Dashboards.
- can be created/edited from the Dashboard Builder, resized, placed, changed in color, etc.
- can be configured to perform a periodic refresh of their data recollecting them from storage/API.
- can be created/connected to Proc.Logic / IoT App.
- can be event driven, so that they are capable to update their data without forcing any refresh to they data.
- can collected interaction from the user to send them on Client-/Server-Side Business Logic.
- can be controlled by other widgets.
- can be controlled by the Proc.Logic / IoT App which can command the widget to show specific data from the storage, specific values, etc., and their combinations (Server-Side Business Logic).
- can be controlled by the Client-Side Business Logic in JavaScript coded in other on in the same widget to send/receive command to show specific data from the storage, specific values, etc., and their combinations, and also some computation, etc.
- etc.



The dashboards can be classified into

- **Passive Dashboards:** showing data taken from **Storage** only, no actions toward Processing Logic (IoT App) node-RED neither on custom JavaScript (ONLY BLUE ARROWS in the above figure):
  - Passive dashboards may have widgets of any kind, and a lot of visualization tools without changing the status of Entities on platform, nor sending commands to the Server Side.
  - Passive dashboards are used to create rendering views of the data in the storage and event driven from their changes with some limited logic pre-coded in the dashboard. For example, a Dashboard with a map and a menu from which the user may decide what is going to be visualized in the page, to browse the data, and see the historical time trend of the time series, etc.
- **Active Dashboards,** are those that show that from the storage and in addition send/receive commands to/from the logic coded somehow (BLUE and GREEN ARROWS in the above figure) and in particular for
  - **Server-Side Business Logic** → logic on Processing Logic (IoT Apps) with Snap4City Dashboard Nodes, which is easier to be programmed begin based on Node-RED visual programming.
  - **Client-Side Business Logic** → logic on JavaScript on specific Dashboard Widgets only for authorized Area Managers developers of Snap4City Platforms. We suggest first prototype by using Server-Side Business Logic, then pass to Client-Side Business Logic in JavaScript. Client-Side Business Logic is coded into the Widgets providing events via the Dashboard Builder in Editing Mode. Client-Side Business Logic may exploit a large number of events provoked by the user on the Dashboard Widgets. **A specific user manual will be accessible soon, in addition to what is published on web pages.**
  - Both kinds of Business Logics may be active on the same Active Dashboard.

**The Active Dashboards** are used to implement Business Intelligence solutions with high interactivity and the possibility of changing the data and the representation of data on the Dashboard dynamically on the basis of the user actions. Examples are:

- the click on some button or widget on Dashboard to activate a computation on Client/Server side. The computation on server side can include the activation of complex Data Analytics to be shown as a result on the same or other dashboard as event drive actions.
- To select some data and perform a query showing them on map and barseries, pie, multi-trend, etc.
- To move a slider and see the light on dashboard changing.
- To filter data from a certain time windows and see the changes also on other data representations.
- Select a time period on a time series and see all the other time trends aligned to the same period.

- Select a PIN on map and see a barseries proving last year average data regarding that element.
- Etc.

#### IV.B.5.b- Summary of Dashboard Widgets' Capabilities for Business Logics

A summary (not all Snap4City Dashboard Widgets are listed in this table)

| Widget name / description                                 | Event Driven | Some Local Interaction | Server-Side Business Logic Node-RED | Client-Side Business Logic (IN = JavaScript) |
|-----------------------------------------------------------|--------------|------------------------|-------------------------------------|----------------------------------------------|
| Single Content                                            | Yes          | --                     | OUT                                 | OUT                                          |
| Speedometer                                               | Yes          | --                     | OUT                                 | OUT                                          |
| Gauge                                                     | Yes          | --                     | OUT                                 | OUT                                          |
| MultiSeries, Time Series, Curved Line, time compare       | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| Time Trend, Time Series                                   | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| Spidernet, Radar                                          | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| BarSeries                                                 | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| Donut, Pie                                                | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| Table                                                     | Yes          | Yes                    | OUT                                 | IN/OUT                                       |
| Device Table                                              | Yes          | Yes                    | IN/OUT                              | IN/OUT                                       |
| Multi Data Map (dashboard Map)                            | Yes          | Yes                    | IN/OUT                              | IN/OUT                                       |
| Selector                                                  | Yes          | Yes                    | --                                  | --                                           |
| Button, Impulse button                                    | Yes          | Yes                    | IN                                  | IN                                           |
| Switch, on/off but.                                       | Yes          | Yes                    | IN/OUT                              | IN/OUT                                       |
| Knob, Dimer                                               | Yes          | Yes                    | IN/OUT                              | IN/OUT                                       |
| Keypad, Num Pad                                           | Yes          | Yes                    | IN                                  | IN                                           |
| External Content                                          | Yes          | Yes                    | IN/OUT                              | IN/OUT                                       |
| Event Driven MyKPI                                        | Yes          | --                     | IN                                  | --                                           |
| Synoptics (see External Content) (read, write, subscribe) | Yes          | Yes                    | IN/OUT                              | --                                           |
| Dashboard Form                                            | Yes          | Yes                    | IN/OUT                              | Possible on Ext.Content                      |
| Speak Synthesis                                           | Yes          | --                     | OUT                                 | Possible on Ext.Content                      |
| D3 charts                                                 | Yes          | Yes                    | OUT                                 | --                                           |

#### Server-Side Business Logic, SSBL:

- **IN** means that the Dashboard Widget may have a counterpart on Proc.Logic and a user action produces an Input event into the Proc.Logic flow, the node in Node-RED has an output from which the event arrives in the JavaScript.
- **OUT** means that the Dashboard Widget may have a counterpart on Proc.Logic as a node in Node-RED having an input to which sending a JSON can produce an OUTPUT on the Dashboard Widget.

#### Client-Side Business Logic, CSBL:

- **IN** means that the Dashboard Widget may have the possibility of caching a user action, and to manage that action with a JavaScript (coded into the Widget More Option, CK Editor part). The JavaScript can produce changes and can sending commands including JSON, to one or more OUT widgets of the same dashboard, and may also open a new Dashboard controlling it.
- **OUT** means that the Dashboard Widget may receive some commands including JSON from others IN Widgets to change its status: what is rendering, the collected and shown data from Smart City API, widget color, position, zoom, content, data, etc., according to the Widget kind and features.

#### IV.B.5.c- *Design process for graphics user interface*

In Snap4City, there is a specific tutorial for the Dashboard development with several examples:

<https://www.snap4city.org/download/video/course/das/>

The **steps in design the Graphic User Interface** would be:

- **Sketch the dashboard** according to the schema reported in the next section. It that should be adopted to design each single Dashboard/view of the solution. The list of graphical widgets available in Snap4City to compose the user interface is accessible at the following link:  
<https://www.snap4city.org/download/video/course/das/>  
For each **data/info to be represented in the Graphic User Interface** one should identify:
  - Preferred kind of data representations: pie, charts, time series, tables, maps, etc.
  - Combination of them, why and how.
  - Level of interaction required by the user.
  - Who will need to access to those dashboards.
  - Who is going to access at the dashboard which provide data and actions in input.
    - Understand if the Dashboard is Active or Passive. Please note that a Dashboard is a View, and each view can be delegated in access to specific users, or user group, or org, etc.
- **Design the interconnections among Dashboards/Views** which are connected each other via Menu or Special internal menus, or buttons or generated from one dash to another.
- Design the user interaction taking into account the capabilities of Dashboard Builder Widgets. Take in mind that some of the actions performed on Widgets can directly perform reaction on other widgets without coding, just configuration.
- **Implement firstly the passive version of the dashboard** and then add Server-Side Business Logic
- Once tested and validated, clone the Dashboard and modify the cloned dashboard to transform it in fully scalable solution with Client-Side Business Logic Active Dashboard.

#### IV.B.5.d- Example of Dashboard Schema

For each Dashboard or View we suggest specifying:

| Name           | Vehicle dashboard                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Aim            | Display vehicle information and measured values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Purpose        | Monitoring                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Status         | Draft                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Missing        | None                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Preferred size | PC                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Style          | PA                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Chat enabled   | No                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Kind           | Active                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Data vs Widget | <p><b>Map Widget</b></p> <ul style="list-style-type: none"> <li>Description: map showing the vehicle position over time</li> <li>Kind: monitoring only</li> <li>Preferred Data representation: map</li> <li>Data: Vehicle.latitude, Vehicle.longitude</li> </ul> <p><b>DataTable Widget</b></p> <ul style="list-style-type: none"> <li>Description: table reporting the vehicle events</li> <li>Kind: monitoring only</li> <li>Preferred Data representation: table</li> <li>Data: VehicleEvent.eventID, VehicleEvent.dateObserved, VehicleEvent.status, VehicleEvent.kind</li> </ul> <p><b>SingleContent Widget</b></p> <ul style="list-style-type: none"> <li>Description: single content showing the total km travelled by the vehicle</li> <li>Kind: Processing Logic / IoT App</li> <li>Preferred Data representation: single number</li> <li>Data: Vehicle.kmTotal</li> </ul> <p><b>Synoptic Widget</b></p> <ul style="list-style-type: none"> <li>Description: battery shaped synoptic to represent the available energy percentage</li> <li>Kind: monitoring only</li> <li>Preferred Data representation: animated synoptic</li> <li>Data: Vehicle.energyLevel</li> </ul> <p><b>Time series Widget</b></p> <ul style="list-style-type: none"> <li>Description: to plot the evolution of the velocity and acceleration values over time</li> <li>Kind: SC Business Logic</li> <li>Preferred Data representation: time series plot</li> </ul> |



|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                   | <ul style="list-style-type: none"> <li>Data: Vehicle.velocity, Vehicle.acceleration</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Client-Side Business Logic</b> | <b>To be developed in JavaScript into the Dashboard Widget</b> <ul style="list-style-type: none"> <li>Identification of the actions performed by the user on the GUI, and for each action what the user is expected to get back</li> <li>Event driven: .....</li> </ul>                                                                                                                                                                                                                             |
| <b>Server-Side Business Logic</b> | <b>To be developed in Processing Logic / IoT App with S4C Dashboard Nodes</b> <ul style="list-style-type: none"> <li>Processing Logic / IoT Application → <ul style="list-style-type: none"> <li>Identification of the actions performed by the user on the GUI, and for each action what the user is expected to get back</li> <li>Event driven: .....</li> </ul> </li> <li>Processing Logic / IoT Application → <ul style="list-style-type: none"> <li>Event driven: .....</li> </ul> </li> </ul> |

As can be seen in the example dashboard schema above, several information must be specified:

- Name:** name or ID of the dashboard
- Mockup:** a graphical example showing the overall appearance of the dashboard. This can be realized using some graphic painting tool (a screenshot of an empty dashboard can be used as background element)
- Aim:** a description of the dashboard
- Purpose:** it can be monitoring, simulation, what-if, data entry, etc. Multiple values are possible.
- Status:** it can be draft, developed, finalized, accepted
- Missing:** in this field list all missing element that should be included in future
- Preferred Size:** specify the preferred viewing size of the dashboard, such as PC, HD, mobile, or an explicit resolution size (row x column)
- Style:** the base style to be used for the dashboard. Available styles include Gea, Balloon, PA, Balloon Dark, etc.
- Chat enabled:** yes or no
- Kind:** passive or active. A passive dashboard show data taken from storage only, without sending actions toward an IoT App; however, passive dashboards may have selectors, maps, etc., and a lot of interactive visualization that do not requires neither changes in the status on server, nor sending commands to the server side. Differently, active dashboards, are those that send or receive commands to/from the server side, via some client-side Business Logic, server-side Business Logic on Processing Logic / IoT Apps, or both.
- Data vs Widget:** for each widget required in the dashboard, some information must be specified according to the following schema:
  - Name: the name of the widget to be used
  - Description: a brief description of the widget and its use
  - Kind: monitoring, Processing Logic / IoT App, or Client-Side business logic (note that, the last two entries characterize an active dashboard)
  - Data: the data the be used in the widget, typically retrieved from some Entity Instance (IoT device). Multiple entries can be accepted.
- Client-Side Business Logic:** to be specified if present
  - Description of the effects: a description of the implemented client-side business logic effects
- Server-Side Business Logic:** to be specified if present
  - IoT App: description of the involved Processing Logic / IoT App
  - Event driven indicates to which events the Processing Logic / IoT App responds.

## IV.C. Development Phases

Development of the Several Aspects includes:

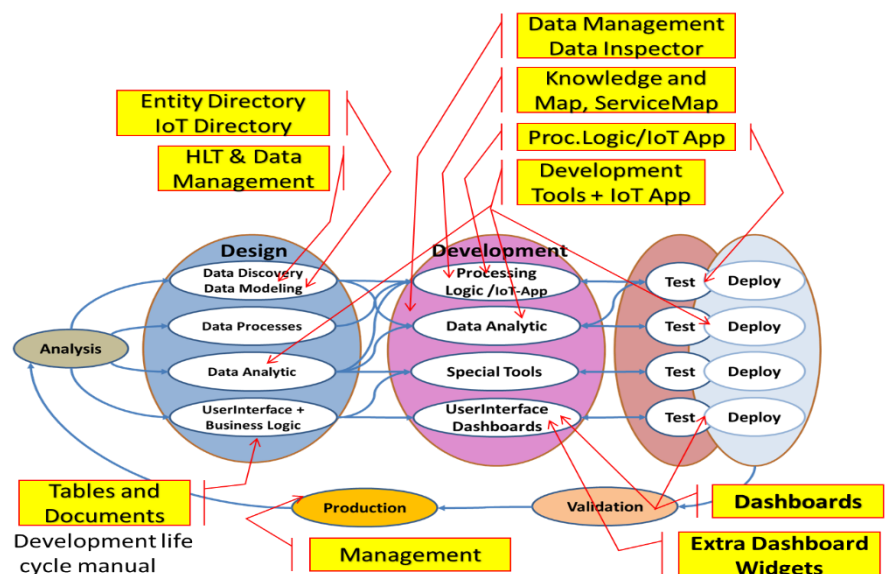
- **Processing Logic (IoT App) Development:** implement Data Ingestion, Data Transformation, Data Analytics, and Server-Side Business Logic as Node-RED flows.
- **Data Analytic Development:** implement complex algorithms to exploit data for producing data descriptive analysis, prescriptions, predictions, early warning, anomaly detections, etc.
- **Special Tool Development:** implementation of special tools to cover new complex data types and create web App which can be integrated into the Dashboards.
- **User Interface as Dashboard Development:** build the user interface using the tools made available from the Snap4City Dashboard Builder including Business Logic (SS and/or CS).

Please note that the development phase is supported by a number of tools in Snap4City. And in most cases the same tools that allow you to develop processes, analytics, and the user interface can be used to test and validate the development you are producing.

We suggest you go on the training course slides to get details about what is just sketched here in this document.

Training Course:

<https://www.snap4city.org/577>



### IV.C.1. Development: Data Processes, aka Processing Logic (IoT Applications)

The development of Data Processes focusses on the functional aspects and includes the implementation of data flow procedures for data: ingestion, transformation, production, publication, load/retrieve, etc. These activities are the classical ETL/ELT of data warehouses and data lakes.

In Snap4City, this activity is strongly simplified since all these aspects are easily developed via **Processing Logic (IoT App)** that is visual programming Node-RED where the usage of JavaScript is quite limited. For massive data ingestion, for example when **large data sources** need to be loaded into the Snap4City platform, it is possible to use dedicated Python processes to perform ETL processes. They can be put in execution as stand-alone processes or just activated and controlled via IoT App which can also send them specific parameters.

So that: **Processing Logic (IoT App)** high flexibility, rapid development any kind of applications, while dedicated Python processes for high performance data ingestion, for example when large amount of data communing from historical repositories has to be loaded in the platform for setup.

**Processing Logic (IoT App) Development** allows exploiting a large collection of macros/nodes (Node-RED

Nodes) and most of them refer to MicroServices which are: the standard nodes provided into the Node-RED editor also called **Processing Logic Editor** (IoT App) in the context of Snap4City. Those provided by Snap4City (more than 180 MicroServices, which can be installed directly from the Node-RED library, repository of Palette); any other MicroServices/Node from other Libraries of Node-RED nodes and from the web since Node-RED is very diffuse.

**Processing Logic (IoT App) of Snap4City** can be used for creating data flow integrating in the same flow multiple activities of Data Ingestion, Data Transformation, Data Analytics and Server-Side Business Logic.

- **Processing Logic = Node-RED + Snap4City Libraries**
- Former: **IoT App = Node-RED + Snap4City Libraries**
- and can be executed on cloud as well as on IoT Edge: Linux, Windows, Arm, AXIS, Raspberry Pi, etc.

**For each Independent Flow of each Processing Logic (IoT App)**, according to the analysis performed to correctly perform this task the developer should provide answer at questions such as:

- Where these processes would be executed? It has to be executed on Cloud Snap4City platform or on some Edge computing of any kind?
- How I have to schedule this process if it not Event Driven? Periodic event?
- Which kind of Processing Logic (IoT App) I need to develop? Which kind of MicroServices I need to have on the Processing Logic node-red among the several possible?
- I have to process historical data or real time data, offline or in real time?
- We need to take them (PULL) or they will arrive (PUSH)?
- Which is the amount of data to be taken? They need to be decoded and how?
- Which are the gateways to be interfaces if any? Which kind of authentication?
- Which are the brokers to be interfaces if any? Which kind of authentication?
- etc.

#### *IV.C.1.a- Capabilities of Proc.Logic / IoT App in Snap4City*

**Processing Logic** allows to develop processes to perform:

- **Data Extraction: Ingestion, gathering, harvesting, grabbing**
  - Several examples are reported in the training course at the following links:  
<https://www.snap4city.org/download/video/course/di/>
  - Node-RED + Snap4City environment provide more than 160 protocols IoT and Industry 4.0, web Scraping, external services, any protocol database, etc.
  - Ingesting data via the development of ETL or Processing Logic (IoT App). They can collect data from several different format sources using several different protocols. FTP, WS, HTTP/HTTPS, etc.
  - Web Scraping tools, to collect data from Web pages and send them to some Processing Logic.
  - ingesting data form excel tables and exploiting some visual tool for data ingestion, which lead to pass data into a dedicated Processing Logic.
  - getting data from Open Data channel as CKAN with some Processing Logic (IoT App).
- **Data Storage/save/load and Retrieval/search: load to storage, retrieve from storage**
  - See <https://www.snap4city.org/download/video/course/iot/>
  - Data access: save/retrieve data, query search on expert system, geo-reverse solution, search on expert system Km4City ontology, etc.
  - **Get data and info from Storage:**
    - Smart City API and Services: transport, parking, POI, MyKPI, personal data, scenarios, etc., via dedicated nodes.
    - Brokers providing data in event driven or the last data, via dedicated nodes.
    - External Services and resources, via dedicated nodes or directly with REST Calls.
    - Several platform managers via API: heatmaps, scenarios, typical time trend, multi series,

calendar, maps, OD, TV Cam, trajectories, BIM, 3D shapes, etc.

- **Save data on platform Storage of:**
  - Entities via Orion brokers NGSI V2, nodes (green node).
  - MyKPI via dedicated nodes.
  - Special HLT and dedicated managers APIs as: heatmaps, scenarios, typical time trend, multi series, calendar, maps, OD, TV Cam, trajectories, BIM, 3D shapes, etc.
- **Data Transform: transcoding, decoding, converting, Production, generation, reformatting, etc.**
  - See <https://www.snap4city.org/download/video/course/iot/>
  - Data Transformation/transcoding: binary, hexadecimal, XML, JSON, String, any format.
  - Also the exploitation of Data Analytics in Python and RStudio.
- **Data publishing: post in channels of any kind, etc.**
  - Integration with any service, servers, provider, gateway, brokers, etc.
  - Integration: CKAN, Web Scraping, FTP, Copernicus satellite, Twitter Vigilance, Workflow OpenMaint, Digital Twin BIMServer, any external service REST Call, etc.
- **Data Analytic management:**
  - Using **Data Analytic nodes** which are instantiated as Container from the Processing Logic: managing Python native, calling and scheduling Python/Rstudio containers as snap4city microservices (predictions, anomaly detection, statistics, etc.).
  - Calling **independent Data Analytics processes** which are executed as services, and can be called as REST Calls. The scheduling and the management of these processes has to be performed by some administrator of the platform, or they are located in third party servers/hosts.
  - Using **Python Node-RED embedded processes**, just executing a Python into a node-red and not into a dedicated container. This approach has a limited amount of memory space and use the same container or VM of the Proc.Logic/IoT App executing the node-RED.
- **Data Interoperability, establishing connections with other services:**
  - From the Snap4City platform, with the aim of
    - **providing data**, the best solution is to start communication from the external process, which will have to use the Smart City API, or the API of the Broker (subscription or query), in authenticated manner.
    - **receiving data in Push** the best and safer solution uses the API of the Broker to send that in push on some broker (e.g., MQTT, ORION), in authenticated manner.
    - **receiving data in Pull** the best solution is to start communication protocol from the Processing Logic calling the REST call of the data provider, in agreement with their authentication model and solution.
  - **The REST Call APIs exposed by Node-RED** are strongly discouraged since they are not protected, not scalable, and directly managed into the Processing Logic. In node-red it is possible by using HTTP blocks/nodes to provide a local API interface to receive from other processes a simple REST Calls to get or provide data.
- **Business Logic (Server-Side) of GUI:**
  - User interaction on Dashboard: get data and messages from the user interface, providing messages to the user (form, buttons, switches, animations, selector, maps, etc.).
  - Custom Widgets: SVG, synoptics, animations, dynamic pins on maps, etc.
  - Generate HTML pages and collect data entry from those HTML pages.
- **Event Management and production:**
  - Create a listener to get event driven data from brokers: MQTT or ORION broker, or other kind of brokers and listeners.
  - Receive events from: Dashboards, changes on MyKPI, changes on Web Sockets, etc.
  - Send/post messages to: Telegram, Twitter, Facebook, SMS, emails, etc.
- **Hardware Specific Devices:**

- Getting local IoT Data (entity instances) from Raspberry Pi, Android, Philips, video wall management, etc., such as: I/O, temperature, CPU workload, connected custom sensors, etc.
- Sending events from the platform to local HW specific devices such as: audio, vibrations, SMS, etc.

**Thus, as a remark, the data ingestion can be performed via:**

- **Brokers:** installing new IoT Devices, registering them on some internal broker, they will send data to Broker which in turn will provide data messages in push directly into the platform storage.
  - Snap4City may provide you entry point for Internal ORION Broker, MQTT Brokers, and on demand other brokers kind.
  - Snap4City can be connected to any Kind of External Broker.
- **Proc.Logic:** Connecting the platform to your data sources by using from Processing Logic as described in the sequel. The Proc.Logic can get data from any kind of source with more than 160 protocols.
  - **You can create your custom data ingestion/loader process.**
- **Dedicated Python process:** Using a Snap4City *FastDataLoader in Python* processes to get data from MQTT brokers, files and databases to send them to the Orion Broker of the platform in high-speed manner. <https://www.snap4city.org/831>
- Using advanced tools such as **Data Table Loader**, **POI Loader**, etc.
- Etc.

#### IV.C.1.b- Examples: Processing Logic / IoT App: typical patterns

In this section, a number of data flow patterns are reported to give at a glance some examples to create your Independent Flows of Processing Logic. The nodes used are those reported before (some of them are specific of Snap4City Library). See also:

- First Node-RED tutorial: <https://nodered.org/docs/tutorials/>
- Using editor: <https://nodered.org/docs/user-guide/editor/>
- Training on the main core nodes: <https://nodered.org/docs/user-guide/nodes>
- Once done we suggest you to pass at the Snap4City Training:
  - <https://www.snap4city.org/download/video/course/iot/>

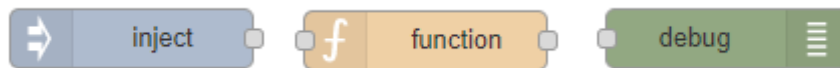
BEFORE PRESENTING THE USAGE OF PROCESSING LOGIC, IOT APP FOR DATA INGESTION, PROCESSING, ETC. WE SHOULD POINT OUT THAT THE BEST SOLUTION TO SEND DATA INTO THE PLATFORM IS BY USING A DIRECT MESSAGE PATCH/POST ON ONE ORION BROKER, JSON NGSI V2. That approach will need for you to be authenticated on the basis of OpenID connect as described in the following in this document. It is not complex, and you will have to do it in any case to perform data access via Smart City API. So that one kind of authentication for both loading and getting data results, in secure, scalable and simple manner. ANY other usage of naïve API and HTTP call to the Proc.Logic and IoT App are viable, but present a number of problems, as described in many points in this document.

- 1) Hello world of node-red, the inject may provide a string to the debug.

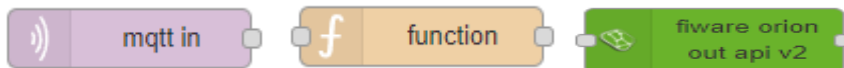


- 2) Hello world of node-red at two steps, the inject provides a push while a JSON is created into the function as `msg.payload = {.....}` and sent/shown to/by the debug.

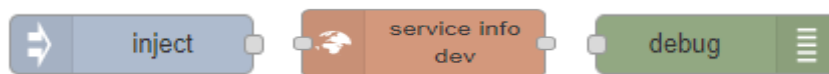




- 3) **SAVE on STORAGE:** Event data reception from an MQTT broker, transformation and send it to the storage pushing data into the Orion Broker V2. A better solution would be to receive data from ORION BROKER NGSI V2 JSON directly on the broker and platform without any need of using a Proc.Logic for data ingestion. This approach is viable if the Entity Instances / Devices are created in advance with some registration procedure.



- 4) **READ from STORAGE:** request on inject of a SURJ to the storage to see data on debug.



- 5) **SEARCH from STORAGE:** Preparation of data request on function, query to the storage and see data result on debug.



- 6) Event data reception from an MQTT broker, transformation to create an Entity Instance from a known Entity Model, debug to see eventual errors, for example if the device is already present (to avoid production of error, one may verify if the Entity Instance is already present by posing a query on the system). A better solution would be to receive data from ORION BROKER directly on it without passing on the Proc.Logic. This approach is viable is the Entity Instances / Devices are already created, that I would say it is quite normal that one prepares the Entity / devices with some registration procedure.



**PLEASE NOTE THAT ALL THAT YOU CAN DO IN MQTT CAN BE DONE IN ORION BROKER NGSI. Moreover, Orion broker is authenticated, in SSO, provides JSON, etc. So that you can have events directly received from Orion Broker and subscription on Orion Broker.**

- 7) Preparation of data parameters on function, request computing Data Analytic, see data result on debug.

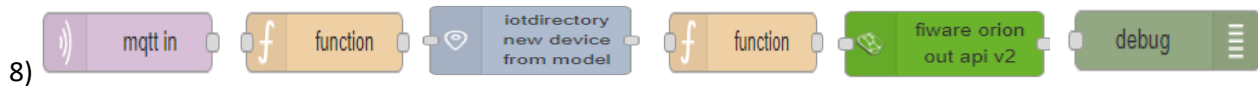


**In the library of nodes accessible and provide on Snap4City Libraries on Node-RED there are more than 150 nodes and microservices. A relevant number of them are provided for interoperability with other platforms and brokers or to make more advanced Proc.Logic programs. For example, a number of nodes for NGSI brokers are also to work on NGSI V1, or V1.1, and to perform query on broker, that in most of the case is not useful since the Broker does not have the ServiceURI and thus the data collected on brokers are difficult to be used.**

#### *IV.C.1.c- Typical strange patterns that may be not efficient in most cases, or wrong:*

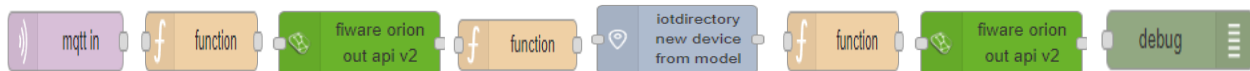
It is possible to create a Proc.Logic to receive data/message from an MQTT broker and their transformation to create an Entity Instance (of a known Entity Model) and use the debug to see eventual ingestion errors.

This approach reported in this case in the following flow, is not optimal since at each new message from MQTT the Entity Directory is queried to see if the Entity has already been created in the past and if not to create it and then pass the data to register the message. In most cases, it is much better to decouple the activity of creating with respect to that of sending message. In fact, this approach would largely reduce the ingestion rate and probably when the Entities are already created would create un-useful workload on Entity Directory (IoT Directory).



Thus, if you have to ingest 1M of messages of a new Entity, you perform 1M of searches to understand if the device exists (all of them, except 1 will fail) and thus you do about 1M of messages on storage. So that, about 2M of actions to succeed 1M.

In most cases, the flow should be designed in the opposite order with this logic: try to send the Entity Message, if it fails than create a new Entity Instance by known model, and if successful send again the Entity Message, or just wait for the new message to save it the first.



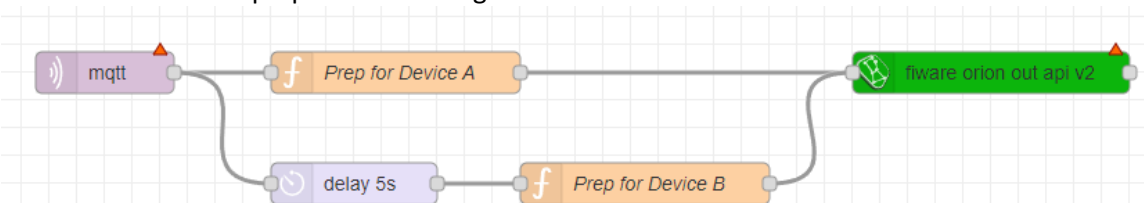
In this second case, if you have to ingest 1M of messages of a new Entity, you perform 1M posts on storage, and you got one fail. You try if the device exists, if not you create. So that about 1M of actions to succeed 1M.

Moreover, according to partner (8) it may happen that at each new message a new device is created, and a data is inserted into the platform since the first Function has a DeviceName depending on the message id, this implies that each new message from MQTT would create a brand new device with only one data inside. In this case, the flow (8) may be totally wrong since the Time Series is never create and thousands of single data are in the storage and they are not connected each other.

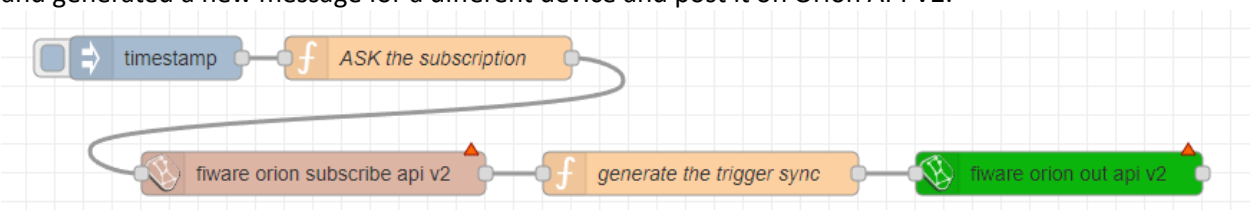
#### IV.C.1.d- Example of Trigger / Synchronization of Device/Entity Data (Pattern)

**If I would like to synchronize a device data A with another B by trigger event, I can do it in several manners.**

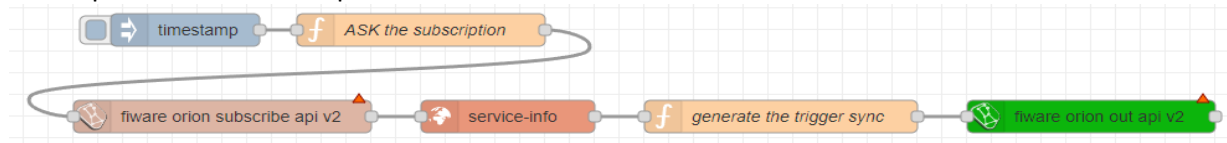
The first case would be the simplest. A triggering message arrives from MQTT event or from some NGSI ORION, or from some MyKPI, from dashboard event button, or email or anything, it does not matter. I can use two functions to prepare the message for A device and B device as follow:



**If the event for triggering is from another device/entity** changed by some action performed posting a data on Orion Broker V2, you can subscribe with the event on the Orion broker by using a specific Node (do it once otherwise you risk receiving many events). Every time the device / entity receives a message you can take it and generated a new message for a different device and post it on Orion API V2.



If you need to verify if the new data has been changed, you can read the last value of recipient Device/entity to compare and decide to update or not:

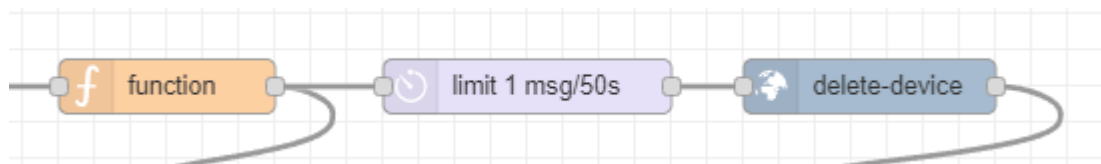


#### IV.C.1.e- Delete Devices

Recently a node to delete devices has been added. It has to be used very carefully since to delete data is always a terrible activity in a big data storage.

The delete of a device is allowed only for the Owner of the device and the root administrator of the platform. The device delete can be also performed from the Entity Directory and now with the Delete-Device node can be performed also from Proc. Logic / IoT App.

The classic pattern is as follows, including preparation, a RATE Limitation avoiding to provide more than one delete message every 50 seconds:



The delete device node needs in input Device ID and Broker ID. All data that you can recover from the Entity Directory.

#### IV.C.1.f- Test and Verify Data Ingestion

In this document we give for granted the data/entity ingestion which can be implemented by the above-mentioned tools for creating Entity Instances, and on them to load Entity Messages. In Snap4City, there is a specific tutorial for the **Processing Logic (IoT App) development** and specific processes for Data Ingestion respectively:

- <https://www.snap4city.org/download/video/course/iot/>
- <https://www.snap4city.org/download/video/course/di/>

Once you have ingested the data, a single message or a set of them attached to a device to create a Time Series, you would need to see if the ingestion has been successfully performed.

**To this end, you can verify and navigated among these elements as in the figure:**

- the arrival of the last Entity Message on the Broker via the Entity/IoT Directory: identify your device and open the [+], click on the PAYLOAD NGSI V2 to see on browser the last data loaded on Broker. Please note the presence of the data on Brokers does not implies that the data is on Storage.

| ID | Name      | Type   | Protocol | Visibility | Format | MAC | Producer | Latitude | Status | Action |
|----|-----------|--------|----------|------------|--------|-----|----------|----------|--------|--------|
| 1  | orionC... | Sensor | ngsi     | PUBLIC     | json   |     |          |          | active | EDIT   |
| 2  | orionC... | Sensor | ngsi     | PUBLIC     | json   |     |          |          | active | EDIT   |

Broker URI: 192.168.1.47  
Kind: sensor  
Device Type: Sensor  
Protocol: ngsi  
Model: PanasonicFatigueTestDevice  
Longitude:  
Device Uri: http://www.disit.org/km4city/resource/iot/orionC...  
Organization: Green4Industry

Broker Port: 8453  
Visibility: public  
Format: json  
MAC:  
Producer:  
Latitude:

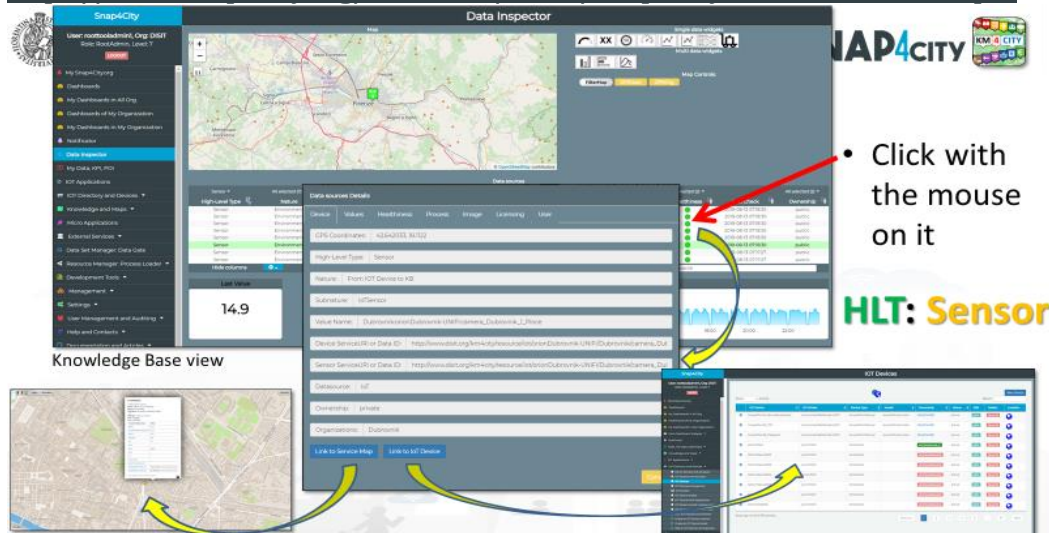
Created on: 2023-04-12 10:21:15

[PAYLOAD NGSI V1](#) [PAYLOAD NGSI V2](#) [NEW DATA IN](#)

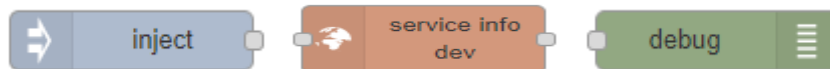
- the arrival of the last Entity Message on the KB indexer via the ServiceMap of the KB or via the

SuperServicemap, knowing the position and the nature/Subnature, or browsing the map, or having the device name you can pose a query (you can identify on Service Map only Public Devices);

- the arrival of data on the storage can be verified by using
  - on Data Inspector as shown in the next figure and described in the <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>



- on some dashboard that you can create.
  - See specific section on how to create a dashboard
- on some IoT App, Proc.Logic you can use a simple flow **READ from STORAGE**: request on inject of a SURF to the storage to see data on debug.



#### IV.C.1.g- Test and Deploy of Proc.Logic / IoT App Node-RED Libraries

As regarding the Test&Deploy, the node-RED environment includes debugging capabilities for the fast prototyping of the flows. The developers can incrementally test the flows step by step in a sandbox and also in the final deploy conditions. Snap4City also supports these passages with the possibility of rebooting, clone, share the Processing Logic (IoT App, Node-RED process) and monitoring its status since they are instanced on Docker Containers.

#### IV.C.2. Development: Data Analytics aka Python and/or Rstudio processes

The development of **Data Analytic** allows to exploit collected and accessible data to produce data hints: descriptive analysis, prescriptions, predictions, early warning, anomaly detections, suggestion, heatmaps, recommendations, decision support, routing, classification, detection, video processing, etc. Most of these processes can use ML, AI, XAI, NLP, operating research, statistic techniques and any kind of libraries.

For each **Data Analytics** one should provide answers to the following questions:

- Which kind of hints I have to develop/produce? Which is my target goal?
- Where are these hints produced?
- How much often I have to execute?
- I have a training and execution phases?
- Which is the best language to exploit certain libraries and AI/XAI models?
- How to assess the quality of what I am going to produce? Which metrics I have to use?

- Etc.

In Snap4City, there is a specific **tutorial for the Data Analytic development** with several examples:

<https://www.snap4city.org/download/video/course/da/>

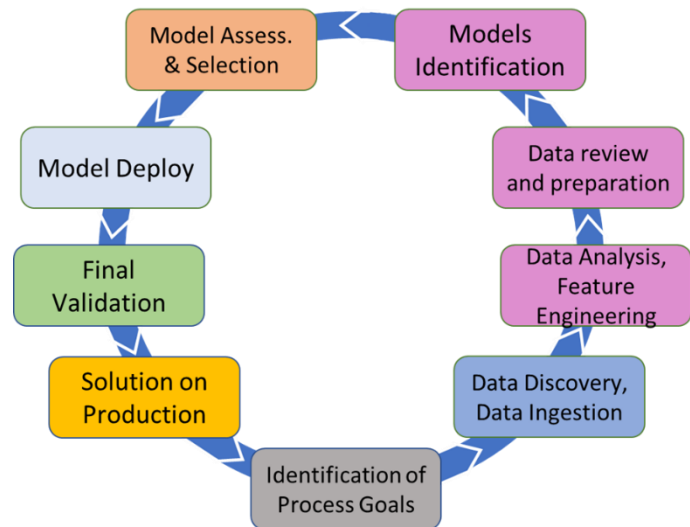
we also suggest reading the Snap4City booklet on Data Analytic solutions.

[https://www.snap4city.org/download/video/DPL\\_SNAP4SOLU.pdf](https://www.snap4city.org/download/video/DPL_SNAP4SOLU.pdf)

**In the following there is a code example in Python.**

#### For Model/Technique Development/testing

- Identification of Process goals and Planning
  - Which goals.
  - How to compute, which language.
  - Which environment, which libraries.
- Data Discovery and Ingestion (from the general life cycle) (as above presented that can be given for granted).
- Data Analysis: feature engineering, feature selection, feature reduction.
- Data review and preparation for the model: management of encoding if necessary, addressing seasonality if needed, data imputation, noise reduction, etc.
- Model Identification and building: ML, AI, etc.
  - Training, Setting ranges and tuning hyperparameters when possible.
- Model Assessment and Selection
  - Assessment on a set of metrics depending on the goals: global relevant and feature assessment.
  - Validation in testing.
  - Global and Local Explanation via Explainable AI, XAI, techniques.
  - Assessing computational costs.
  - Impact Assessment, Ethic Assessment and incidental findings.
- Model Deploy and Final Validation
  - Optimization of computation cost for features, if needed reiterate
- Solution on Production



**Figure 19a: Data analytic, machine learning, AI, XAI life cycle**

The developed DA processes may be activated by a Processing Logic (IoT App) which also passes some parameters for the computations. Classical parameters are the references (ServiceURI) of the data and context on which the DA has to be applied. The results can be directly saved into the storage from the DA or can be passed back to the calling Processing Logic (IoT App) for further processing. The Processing Logic (IoT App) management of DA is strongly appreciated by data scientist since it is very useful when several different learning instances need to be launched in parallel to perform DA Model selection and tuning by using different parameters.

#### IV.C.2.a- Accessing data from Data Analytics Processes

The data analytics processes can access to the information on Snap4City Storage by using REST API Call at Smart City API as described in the following in section IV.C.6, including the needs of performing the



authentication. This means that the Data Analytics processes can only access to the Entities Devices and Messages for which they have been authorized in access via Delegation.

The data analytics processes can also send data to the platform. The best approach is to send that to the corresponding NGSI V2 brokers via API as described in Section IV.C.6, via PATCH as described.

Please remind that to use the private data out of the platform IS NOT GDPR COMPLIANT.

#### IV.C.2.b- *Exploit Data Analytics (Python and/or Rstudio) from Proc.Logic/Node-RED*

**How to proceed:** In Snap4City, this activity is strongly simplified since the Data Analytic can

- **Develop the Data Analytic** on local computer/workstations or on servers by using Python or Rstudio:
  - Snap4City.org provides an online development user interface for Python (Jupyter Hub) and R (RStudio Server).
  - From developed R/PY code you can directly access to Data from Storage performing REST Call API queries may be using some credential or access token to be authorized.
  - From developed R/PY code you can directly send data into Storage sending them as Entity Instances and related Entity Messages (IoT Device messages) into one of the references Brokers at disposal.
  - See example: <https://www.snap4city.org/645>
  - Python usage of API: <https://www.snap4city.org/829>
  - see also in the following the section on authentication and a section on Python access to API
- **load Python or Rstudio on MicroService Node into Processing Logic (IoT App)** to see them converted / deployed in Container on Cloud.
  - bring the Python or Rstudio code into a container conforming the development according to Snap4City guidelines on Data Analytic development.
  - Preparation of data parameters on function to send them on Data Analytic container, JSON to request computing Data Analytic (Python or Rstudio using different Nodes), see data result on debug. The Container of Data Analytic can pass back results and errors see example on the training course: <https://www.snap4city.org/download/video/course/da/>



- **Use the Processing Logic (IoT App) to pass parameters** to Python or Rstudio processes of Data Analytics and get results back into the Processing Logic (IoT App) in JSON, or get errors into the Processing Logic (IoT App) also in JSON.
  - Please note that the Python and Rstudio processes in Container can directly access to the Smart City API. So that if they are designed in parametric manner (for example to work on the basis of data collected from a set of SURTI, and in a time interval and these parameters are passed in the JSON input), the Data analytic node would be capable to provide Data Analytics as a Services on different data every time and automatically collecting the data (even big data amount) via a direct call of the Smart City API, using access token as described in the sequel.
- The user can also develop some Special Tools for implementing specific solutions providing their own visual interface, which may be included into dashboards as an **External Service** providing an HTTP URL.

Rstudio and Python data analytics processes may include conceptually any kind of libraries for ML, AI, operative research, simulation, etc. The Development environment may be configured to allow at the single operators to load their own preferred libraries. On the other hand, when the process is adopted to produce a contains as in the next figure, the container has to include the library used in the code. This can be

performed by requesting a specific image to the platform manager and indicating the library you would like to have on Container executions.

In the following there is a code example in Python.

In Snap4City, there is a specific **tutorial for the Data Analytic development** with several examples:

<https://www.snap4city.org/download/video/course/da/>

#### *IV.C.2.c- Export data from storage for different purposes*

**Please remind that to use the private data out of the platform IS NOT GDPR COMPLIANT.**

**This means that, the fact that you are authorized to ACCESS at the data does not means that you have been authorized to Download, and neither that you have been authorized to redistribute them to third party, etc.**

For this reason, the platform allows you do create a process for downloading data and put those data into a file. This process can be performed in two manners:

- Developing a proc.Logic / IoT App for the purpose
- Developing a Python process to access at the data via API

In both cases, there are some limitations on the amount of data which can be downloaded in one API call.

### IV.C.3. Development: User Interface as Dashboard

The design and development of the smart application Graphic User Interface, GUI, means to develop views which put in connection the back-office data with some visual representation and provide to the user interactive elements to change the observed view or go for another.

In Snap4City, the views are implemented as Dashboards which are composed by graphical widgets connecting storage on the basis of the Entity Instances, Processing Logic (IoT App) data/nodes in stream, Synoptics, External Services, and Brokers [Dashboards2019]. Widgets can be maps, time trends, chords, spidernet, barseries, tables, buttons, animated elements, sliders, etc., from libraries as D3 and Highcharts or custom Synoptics widgets by using an SVG-based integrated tool and templates. Moreover, the Dashboard may dress different styles / themes in their rendering and the designer / developer can select them soon or later in the creation process.



Figure 19b: Several Kinds of Dashboards, examples of themes.

The powerful of the user interface is grounded on the possibility of easily connecting the graphic widgets with the Entity Instances, and with the possibility of gluing them in smart interactive manner, keeping human in the loop (see Figure 19c). For example: selecting an area or clicking on a Pin/service on map and connecting related data to widgets with a pie and to the time series, or producing some computation such as the average values, the max or other, etc. To this end, the User Interface needs to provide some business logic which can be on server-side (formalized in Processing Logic/IoT App, Node-RED) to serve all the users at the same time, and client-side for evolving user interface behavior on each client device autonomously. Client devices are typically a browser but could be also a mobile app.

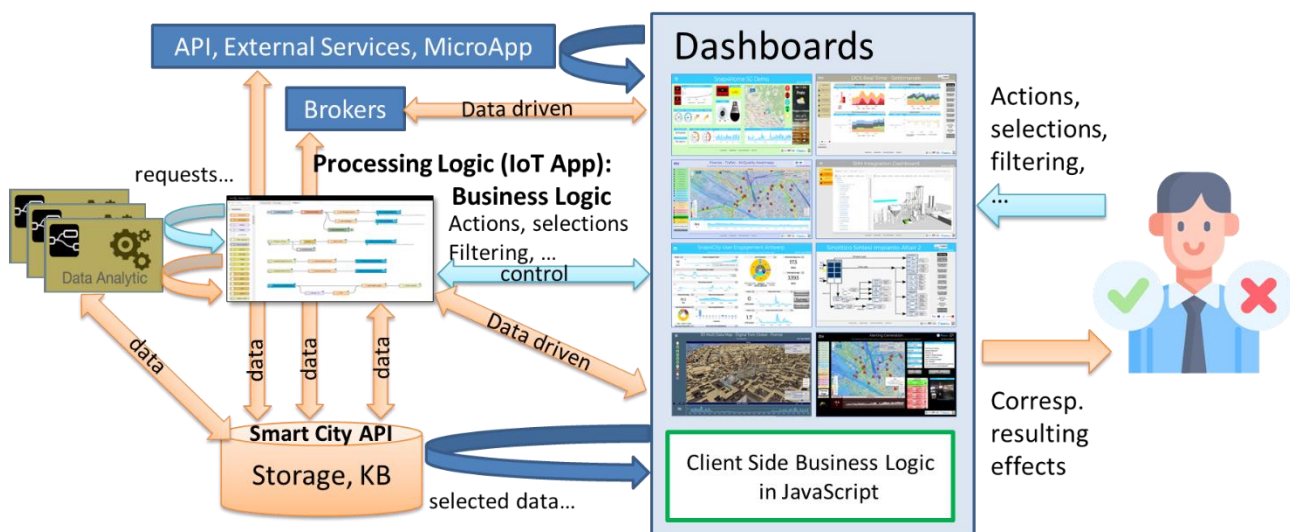


Figure 19c: Relationships among storage, Processing Logic (IoT App), and Dashboards with server-side and client-side business logics.

According to **SMADe-Ic**, the design of views starts with the production of Dashboards' mockups. For each Dashboard a set of suggested information is requested such as: aim, status, target device, GUI style, list of widgets and their description, and the business logic kind.

The **Dashboards are called passive** when the data represented come only from storage without any event driven connection from platform to dashboard and vice versa, for business logic or for just presenting/producing data in real time. **Active views/dashboards** are those that provide those connections. They are initially developed as passive Dashboards in first sprints and get smarter and smarter in successive sprints of the life cycle. Moreover, for fast prototyping the server-side business logic is developed by using IoT App Node-RED, which provides a set of nodes corresponding to connect graphical widgets on dashboards (via secure WebSocket) [MicroServices2019] (thus, overcoming the limitation of native dashboards of Node-RED). This approach is very effective and viable for fast prototyping and to realize strictly synchronized smart applications in which all the actions on web interface are shared with all the users (typically limited) as happen in IoT world. For example, the monitoring panel of an industrial plant should present the same data to all the users connected to it.

On the other hand, when the users expect to play with the data for some business intelligence, their experience and the evolving data represented in the interface according to their activities is going to change. Those aspects are personal, and context based as one expect to have on a smart application, leading to an evolving user interface that should have a client-side business logic. In Snap4City, the development of client-side business logic can be implemented by adding JavaScript functions attached to the graphic widgets call back actions, which can perform actions on other widget and on the platform, such as a REST Call to API. Note also that, typically IoT dashboards are implemented as single web page, however it is also possible to link multiple dashboards so to have different interfaces that the user can navigate. For example, a main dashboard could list some devices, then by clicking on one of such devices a new dashboard is open showing details on the specific device. This can be achieved exploiting the client-side business logic by using specific JavaScript function calls able to both open a new dashboard and possibly send some data.

According to **SMADe-Ic**, IoT App for data ingestion, transformation, etc., and those for business logic could be developed by different developers. On one hand, those functionalities could be located on different flows of the same IoT App, even if it is not suggested. On the other hand, when event-driven applications are developed, the integration of flows for data streams processing and that of business logic may be almost mandatory. A way to allocate those flows in different IoT Apps could be to pass events from two IoT Apps passing messages via one of the platform brokers (NGSI Orion Broker in Snap4City).

To develop this phase one has to follow the Dashboard descriptions performed in the design phase. And at the same time has to answer at questions such as:

- Does the user interface have to provide some dynamic changes on the basis of the user actions? Which kind of changes?
- How many users are using it?
- Is it a view for control room and decision makers or a business intelligence tool for playing with data and solutions?
- Etc.

**How to proceed:** The developers on Snap4City can visually create dashboards with a drag and drop approach by using Dashboard Builder which is assisted by a number of tools shortly described in the sequel.

The main concepts are based on the fact that Dashboards:

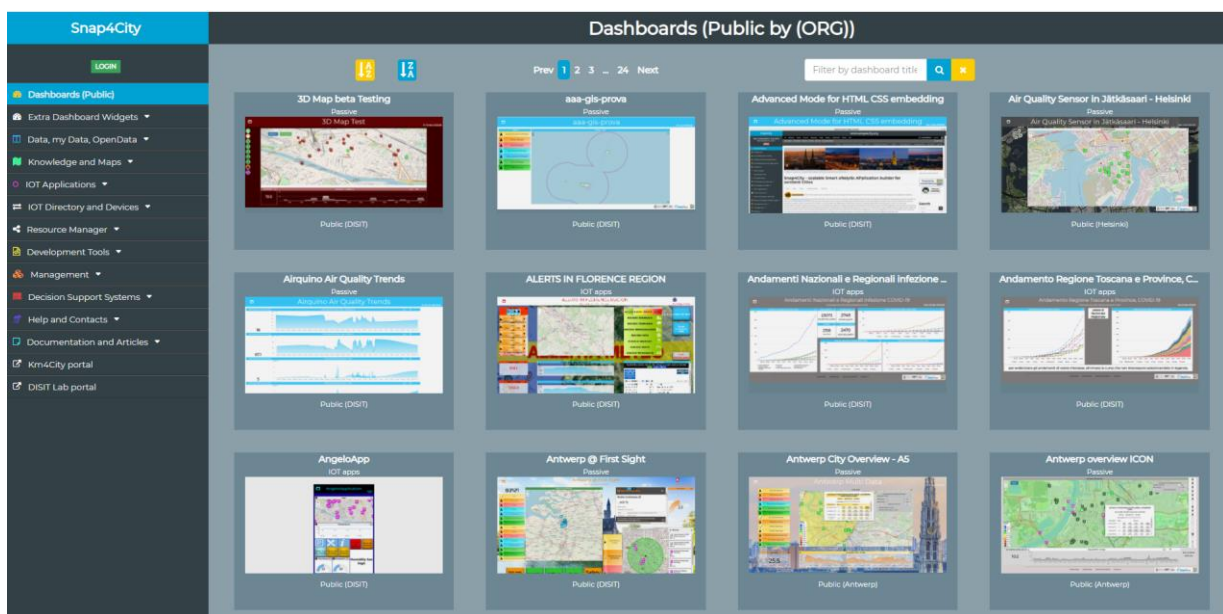
- can be complex web pages for providing information and rendering and ALSO for getting data and interaction from the users.
- are built for composition of a set of Widgets.
- can communicate with Server-Side Business Logic.



- can host internally server-side business logic.
- embed any other frame.
- can be connected each other.

The Snap4City **Dashboard system** is used in several cities and projects and can be accessed from:

- SEE several examples of dashboards from Technical overview.
  - <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>
- Scenarios: <https://www.snap4city.org/4>
- **A large List of Public Dashboards on Snap4City.org** (many others are in other installations of Snap4City, and most of them are not public):  
<https://www.snap4city.org/dashboardSmartCity/management/dashboards.php>



**Snap4City Dashboard system is capable to:**

- show dashboards on the web browser in an H24/7 modality;
- show data on widgets according to several graphic paradigms (tables, graphs, histograms, maps, Kiviart charts (spider net), lists, tv camera, heatmaps, weather, critical city events, 3D, D3.JS, etc.) with a level of interactivity and animation;
- use the Arabic language and Hijri date format and conversions;
- show data on autonomous and connected/ synchronized widgets;
- collect, show, and keep data updated on the screen with an automated refresh for each view, and real-time data according to the even-driven paradigm;
- show data both real-time and historical, allowing the drill down on time, space and relationships among data and city entities;
- provide a solution to have dashboards connected with IoT App and platform by using end-2-end secure connections based on Web Socker secure, TLS, HTTPS;
- collect and show data coming from different big data and classic data sources (SQL, NoSQL, RDF, P2P, API, SOLR, Elastic Search, etc.) also in aggregated manner;
- allow to customize Widgets as Synoptics and also exploiting graphic libraries;
- work with a large amount of data providing high performances, such as short response time;



- compute alarms, and provide support by a flexible notification system capable to send alerts, activating tickets for maintenance, automating actuators, post on social, etc.;
- provide actuators widgets together with showing graphs, and capable to act on IoT Devices;
- provide support for collaborative production of dashboards and co-working;
- provide support for embedding dashboards into third-party web pages;
- provide data engine for collecting connection response time on different protocols, and for verifying the consistency of web pages via HTTPS;
- allow cloning dashboards;
- allow giving access to the dashboards to other users;
- integrating with Proc.Logic / IoT Apps by managing real-time data and connecting its actuators to real-time IoT Apps, thus having in Proc.Logic / IoT App the Server-Side Business logic of one or more dashboards;
- integrate dashboards in more complex dashboard systems;
- script business logic of the Dashboard into the dashboard in JavaScript (Client-Side Business Logic) and/or into Proc.Logic / IoT App;
- Control Video Wall configuration from the Dashboard and/or IoT App Business logic;
- support authentication and authorization with the most general approaches such as LDAP, and SSO;
- collect and get data from batch resources and in real-time, using a large range of protocols and formats;
- Associate them a dashboard specific menus;
- Connect to each dashboard a dedicated chat room for discussing problems and events.
- Etc.

Dashboards may be single or connected. They are typically not a single view, but a view of a set. From the main Dashboard you may need to jump to other views/Dashboards. See the following example of the **Smart City Control Room, SCCR**, of **Florence Metropolitan City** (since 2017) which has more than 1.5 million inhabitants, and >14 million Tourists per year, plus students and commuters. The figure shows the main dashboards used by the Florence Mayor (Dario Nardella) and the second-level dashboards. The third and fourth levels are present as well. <https://www.snap4city.org/525>

Control room with control video wall: <https://www.snap4city.org/621>

Florence Control Room: <https://www.snap4city.org/531>

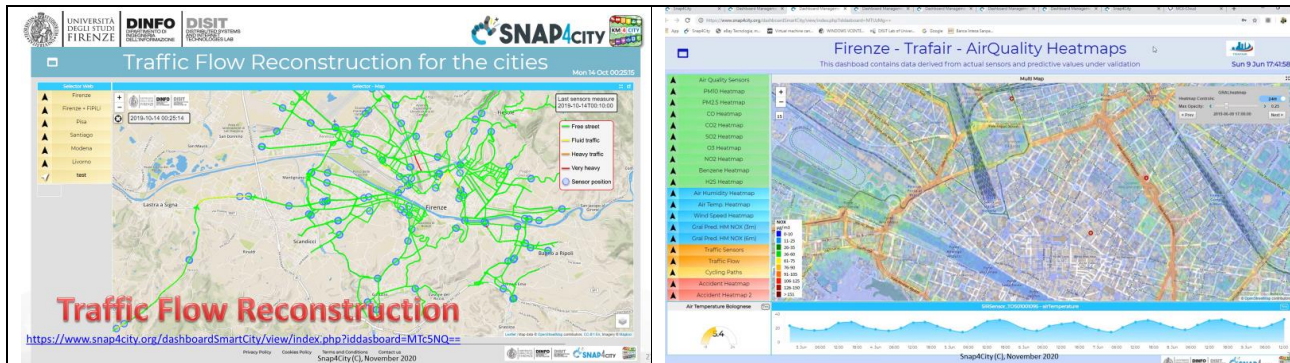


Dashboards may show data coming from Big Data stores, IoT Apps and other sources/databases. They are produced in real-time and may show real-time event-driven data for the decision-makers, officials, city users, totem, operators, fire brigade, emergency, civil protection, police, operators, leaders, etc., according to a controlled and secure connection on HTTPS, secure WebSocket and GDPR compliant environment. They allow representing and managing critical events, receiving notifications, drilling down on data, opening live

chats for problem-solving, acting in correspondence of alarms by an intelligent monitoring, defining workflows, performing simulation, and **What-IF analysis**.

For **what-if analysis** see:

<https://www.snap4city.org/download/video/course/da/>



In Snap4City, there is a specific **tutorial for the Dashboard development** with several examples:

- <https://www.snap4city.org/download/video/course/das/>

#### IV.C.3.a- The List of Dashboard Widgets, kinds of widgets

The list of graphical widgets available in Snap4City to compose the user interface is accessible at the following link: <https://www.snap4city.org/download/video/course/das/>

A short summary of Dashboard Widgets is also provided in:

- **TECHNICAL OVERVIEW:** <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>

**The dashboard widgets can be:**

- Generated from IoT App / Proc.Logic as node-red nodes of Snap4City libraries. In this case they are gates for creating Server-Side Business Logic. In most cases, a widget created in this manner does not have client-side business logic.
- Generated Manually from the Dashboard builder.
  - Certain widgets may have in More Options the possibility for providing JavaScript for specifying the Client-Side Business Logic.
- Generated from Dashboard Wizard and then connected to some Node into the IoT App / Proc. Logic
  - not all widgets created from the Dashboard builder or wizard can be connected, the MultiData map can be connected, thus created by wizard and connected to the IoT App / Proc. Logic.
  - Certain widgets may have in More Options the possibility for providing JavaScript for specifying the Client-Side Business Logic.

### IV.C.3.b- The Dashboard Builder

Snap4City **Dashboard Builder** enables users of any role in the platform to create and manipulate their Dashboards of any kind, passing from simple data visualization tools to business intelligence tools, to control room dashboards and interactive systems and synoptics, including full Digital Twin solutions with sophisticated integrated 3D representation at low cost. See details on dashboard from the training course part 2: <https://www.snap4city.org/download/video/course/das/> [Industry4.0-2020], [Dashboards2019], [DashboardProduction2020], [ChemicalPlant2021].

**Dashboard Builder** allows the creation of Dashboards:

- **Manual selection and composition of Widgets** on Dashboards. With the Dashboard builder the developer can arrange the widget in the frame and may change some setting using the so called More Options of each Widget.
- **automatically** using the **Dashboard Wizard** to simplify the production and the connection of **data to widgets** according to models/templates. Once creates, they can be manually edited for defining and changing all detailed aspects. **Dashboard Wizard** is an expert system for immediate matching HLT data vs graphics representation for creating Dashboards by rendering and acting on data with a large range of graphics widgets.



- **widgets** may
  - be **used in editing mode** with the **More Options** can be accessed by selecting it in the widget contextual menu that can be open by clicking on the (☐) icon on the top-right corner of the widget.
    - **More Options** allows to customize widget in deep for: data source, size, colors, shape, font, strings UTF-8 also Arabic, dates, left/right time series, staked/non staked, by value / by type, comparison or not, text change, data order change, etc.
  - have coded **business intelligence**.
    - **More Option** may include **CK Editor** for embedding Client-Side Business Logic, only for authorized Area Managers. Please ask to your administrator.
    - **May be connected to Proc.Logic (Server-Side Business Logic)** having a counterpart into the Node-RED editor, and even produced by the Proc.Logic when a Dashboard Node is

placed declaring the Dashboard on which one would like to work with. This is the approach by which one Dashboard may request the activation of Data Analytics or other server-side complex data processing (in batch or event driven), for example.

- be **Customized** for creating new widgets and **synoptics** using visual tools and templates: for real time rendering data on graphical scenographic tools, and for graphic interaction on the systems from dashboard to actuators end-to-end secure connection.
- exploit **External Service** for integrating external tools via Processing Logic and/or via IFRAME into External Content Widget.
- **styling using a large set of styles**, which can be customized and enlarged by following a simple tutorial. The passage from one style to another is immediate, and the default style can be chosen for each of your Dashboard.
- **Creating/activating menu and connections among dashboards and other pages**, or exploiting the organization menu on the upper left corner
- **Connecting a Chat of the Dashboard** for distributed Control Room and situation rooms discussion.
- Etc.

#### IV.C.4. Development: Server-Side Business Logic, as Processing Logic

This approach is a prerogative of Snap4City development environment. In Snap4City, it is possible to have one Proc.Logic (IoT App) referring to multiple Dashboards, and one Dashboard referring multiple Processing Logic (IoT Apps). For this purpose, the Snap4City libraries on Node-RED includes a number of dashboard nodes as reported in **Figure 19c**. They are used in the Proc.Logic / Node-RED and at the deploy they are also created into the Dashboard. How to work with widget/nodes is reported in the online help into the Node-RED visual programming environment.

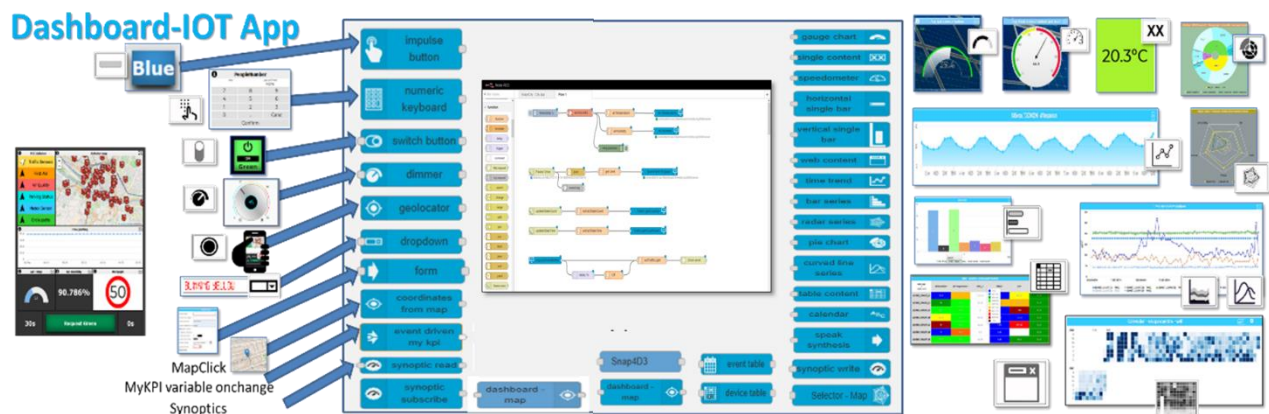
According to the training <https://www.snap4city.org/download/video/course/das/>

Let see a 1:1 relationship from Proc.Logic and Dashboard:

- Any Action performed on Dashboard is provided to the Proc.Logic, which may produce reactions on Dashboard.
- The context of Proc.Logic $\leftrightarrow$ Dashboard is a singleton, thus any user connected to the Dashboard will observe the evolutions performed. So that all the users will see the same story and view, even if they are authenticated and authorized by using different credentials, and they are on multiple dislocated terminals.
- This is good for control rooms, and single/few users prototypes.

This activity is performed by using a number of Dashboard Widgets which have the counterpart on the Proc.Logic (IoT App) Node-RED as above summarized.



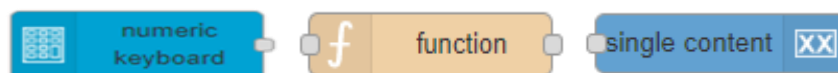


**Figure 19c: Dashboards with Widgets and their counterpart on Proc.Logic for implementing Server-Side Business Logic widgets/nodes: in (left), out (right) and in/out (left and right).**

According to the training <https://www.snap4city.org/download/video/course/das/> some of widgets are capable to:

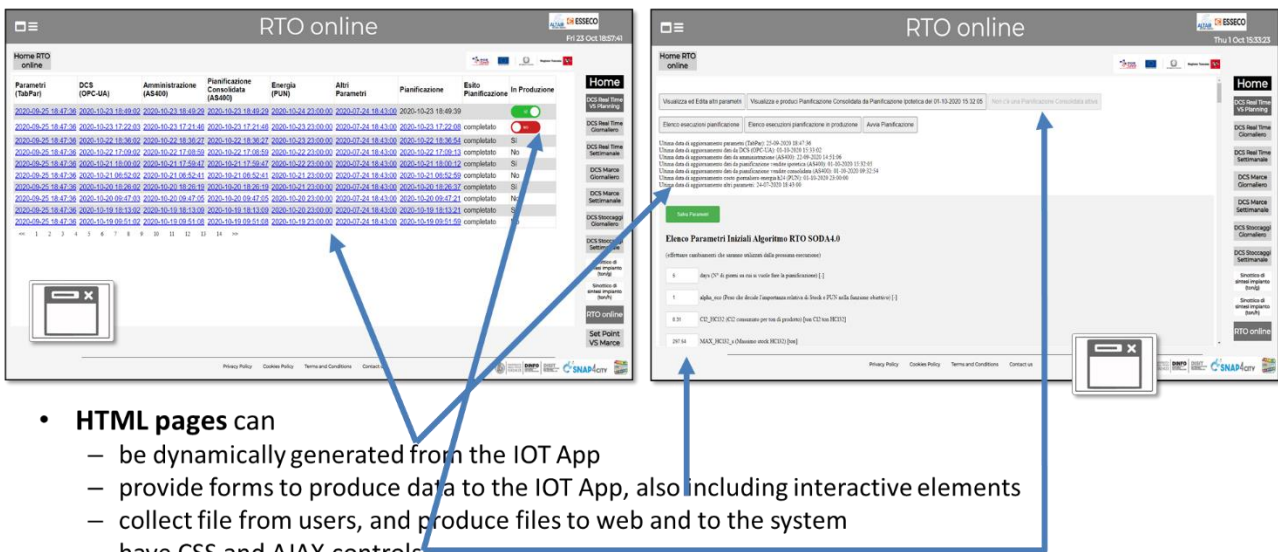
- Send information and commands to the Dashboard Widget, for example for an action produced by the users. (**IN widget/node**)
- Receive information and commands from the Dashboard Widget, for example presenting a dashboard change to the users. (**OUT widget/node**).
- Send/receive information and commands to/from the Dashboard Widget, for example for collecting users' actions and presenting a change to the users on the same widget (**in/out widget/node**).

The typical pattern can be IN widget receiving data from dashboard, function, then an OUT Widget sending data on dashboard:



On Server-Side (into Proc.Logic) the developer can even create dynamically HTML pages of any kind and embed them into a Dashboard Widget as external content. A mix of in, out, and in/out widgets can also be included in the same dashboard. This approach, shown in **Figure 19d**, is very powerful and presents the behavior of Server-Side Business Logic for control room described above.





- **HTML pages can**
  - be dynamically generated from the IOT App
  - provide forms to produce data to the IOT App, also including interactive elements
  - collect file from users, and produce files to web and to the system
  - have CSS and AJAX controls

**Figure 19d: Dashboards with External Content Widgets hosting HTML page produced by Server-Side Business Logic, and recollecting/presenting data to the user.**

Note that, dynamic HTML pages can include in/out callbacks from/to Proc.Logic (IoT App) using web sockets. In this way the HTML page can exchange information with an IoT App to interactively update some visualizations: being such data get or sent to server-side business logic (implemented in Proc.Logic (IoT App) Node-RED) changes in the visualization will be reflected to all the users viewing the same dashboard. Differently, if the HTML page includes some JavaScript client-side logic (such as a table with sortable columns) the actions performed by one user will not be reflected to other user accessing the same dashboard (see more about this in the next section).

#### IV.C.4.a- Examples of Proc.Logic Event Driven Business Logic Nodes / Patterns

The Proc.Logic can be used to create event driven processing logic. To this end a number of Node-RED nodes can produce a message into the Proc.Logic for activating the internal processing, while other nodes can produce events in other contexts such as other Brokers, Proc.Logic, Dashboards, etc.

**For example, Event Driven Inputs Widgets are:**

- **inject** A periodic event can be programmed.
- **mqtt in** An MQTT event on some entity arriving from an MQTT Broker
- **orion-subscribe-api-v2** A NGSI V2 event from some ORION Broker (internal broker of the platform)

Other Brokers can be used such as: STOMP, AMQP, WS, RSS, email, twitter, SIGFOX, M2M, etc.

Other events can be received by implementing a HTTP server / API by using HTTP Node and thus produced for the single Proc.Logic. We strongly discourage this approach which is not safe and neither scalable. Similar statements for other Node-RED nodes such as: TCP.

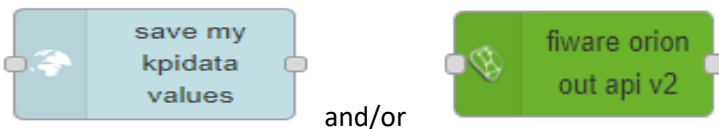
### From Dashboards:

- An event from a change on some Snap4City MyKPI variable.
- An event from a Snap4City Synoptic, a click of a user
- An event of click on snap4city Multi Data Map.

Please see table on **Section IV.B.5.b** for the full list of event-driven Dashboard Widgets.

A large number of Dashboard widgets can be used for rendering event driven data updates on Dashboards.

Another Proc.Logic can communicate with the former by using events on Brokers or on MyKPI. Sending messages on Entity Instances / IoT Devices via Orion Broker, as well as by saving MyKPI data values:



### IV.C.4.b- Dashboard Widgets: gates for Server-Side Business Logic

The list of graphical widgets available in Snap4City to compose the user interface is accessible at the following link: <https://www.snap4city.org/download/video/course/das/>

The following Snap4City nodes for Node-RED to implement Proc.Logic/IoT App are created into the Node-RED and into the configuration the user has to indicate the dashboard in which they have to appear. As alternative the user may also create a new Dashboards directly from the IoT App, and this Dashboard to push the new widgets having a counterpart into the IoT App itself.

Most of the following nodes communicate with the Dashboards by means of WebSocket.

All the users accessing to the same Dashboard will produce data, events on the same IoT App, and the data provided by the same IoT App is going to send the data on all the users connected on the same Dashboard and will see the changes all together, all the same time. This is a Control Room approach, as singleton and collaborative work on it. For other interaction models see later Client-Side Business Logic.

Each node has some help directly accessible into the Node-RED editor with precise JSON details of input and outputs messages.

Each counterpart widget presents a menu item called MoreOptions in which several detailed configuration can be defined on how to show the data which are provided from the Proc.Logic. Please used it to better results.

Another summary of Dashboard Widgets is also provided in:

- TECHNICAL OVERVIEW:** <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>

**NOT ALL Dashboard Widgets of Snap4City are Listed in this table, which presents ONLY THOSE which have a counterpart on Proc.Logic /IoT App Node-RED!**

| Node on Snap4City | name                   | Description WIDGET                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | Gauge                  | Receive a JSON with <b>one</b> ServiceURI and/or value to be shown on gauge. You do not need to send data, only the ServiceURI, the data are automatically collected and updated by the Widget. You can dynamically change the ServiceURI.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|                   | Speedometer            | Receive a JSON with <b>one</b> ServiceURI and/or value to be shown on speedometer. You do not need to send data, only the ServiceURI, the data are automatically collected and updated by the Widget. You can dynamically change the ServiceURI.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|                   | BarSeries              | Receive a JSON with a <b>set</b> of ServiceURI and/or values to be shown on BarSeries. You do not need to send data, only the ServiceURI, the data are automatically collected and updated by the Widget. You can dynamically change the ServiceURI.                                                                                                                                                                                                                                                                                                                                                                                                                              |
|                   | Switch, on/off but.    | <ol style="list-style-type: none"> <li>1) Get a status from input PIN of the node in the flow to change its status and show the status changed on dashboards.</li> <li>2) Receive a change from the dashboards and communicate it into the flow on Output PIN of the node.</li> </ol>                                                                                                                                                                                                                                                                                                                                                                                             |
|                   | Knob, Dimer            | <ol style="list-style-type: none"> <li>1) Get a status from input PIN of the node in the flow to change its status and show the status changed on dashboards.</li> <li>2) Receive a change from the dashboards and communicate it into the flow on Output PIN of the node.</li> </ol>                                                                                                                                                                                                                                                                                                                                                                                             |
|                   | Button, Impulse button | The widget on dashboard provides an impulse button and the user acting on it sends the event in the flow from the output PIN on this node.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|                   | Keypad, Numeric Pad    | Numeric Keypad widget on dashboards which sends an event into the flow from the output Pin of this node regarding the value imposed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|                   | Device Table           | <ol style="list-style-type: none"> <li>1) Receive a JSON with a <b>set</b> of ServiceURI to be shown on table, plus eventual controls for interacting with. You do not need to send data, only the ServiceURI, the data are automatically collected and updated by the Widget. You can dynamically change the SURI. You can describe which data have to be shown, names, order, etc. and you can add custom icon buttons per line of the table.</li> <li>2) The user may perform action on device table widget which are sent to the flow from the Output PIN of this node.</li> </ol> For details See: <a href="https://www.snap4city.org/809">https://www.snap4city.org/809</a> |
|                   | dropdown               | <p>A widget which presents a selector to the users on the basis of a configuration defined into the node and dynamically changed by sending a JSON in the input PIN of the node.</p> <p>The selection performed produces a message from the output PIN of this node in the flow.</p>                                                                                                                                                                                                                                                                                                                                                                                              |
|                   | Dashboard Form         | <p>A widget which presents a FORM to request data (integrate, on/off, date time, etc.) to the users on the basis of a configuration defined into the node and dynamically changed by sending a JSON in the input PIN of the node.</p> <p>The insert data produced by the users produce a message from the output PIN of this node in the flow.</p>                                                                                                                                                                                                                                                                                                                                |

|  |                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Single Content                                      | A widget which may receive and show on dashboard single data (integer, string, time, etc.) and also HTML with some CSS formats.                                                                                                                                                                                                                                                                                                                                                                                                  |
|  | External Content                                    | A widget for showing some HTML page on dashboard. The HTML page can be changed dynamically.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|  | Spidernet, Radar                                    | The node receives a JSON with a <b>set</b> of ServiceURI and/or values to be shown on Spidernet, Radar widget on Dashboard.                                                                                                                                                                                                                                                                                                                                                                                                      |
|  | Donut, Pie                                          | Receive a JSON with a <b>set</b> of ServiceURI and/or values to be shown on Donut, Pie widget on Dashboard                                                                                                                                                                                                                                                                                                                                                                                                                       |
|  | BarSeries                                           | Receive a JSON with a set of ServiceURI and/or value to be shown on BarSeries widget on Dashboard.<br>The bars can be vertical, horizontal, staked non staked, etc. See MoreOptions on the Dashboards builder for changing the show modality.                                                                                                                                                                                                                                                                                    |
|  | Table content                                       | Receive a JSON with a <b>set</b> of ServiceURI and/or values to be shown on table on Dashboard. The table can have colored cells according to colormap standard.<br>it is less flexible of device table, but simpler to be used.                                                                                                                                                                                                                                                                                                 |
|  | Time Trend, Time Series                             | Receive a JSON with <b>one</b> ServiceURI and/or value to be shown on Time Trend, Time Series widget on Dashboard                                                                                                                                                                                                                                                                                                                                                                                                                |
|  | MultiSeries, Time Series, Curved Line, time compare | Receive a JSON with a <b>set</b> of ServiceURI and/or values to be shown on MultiSeries, Time Series, Curved Line, time compare widget on Dashboard. The trends can be staked / non staked, shaded / non shaded, double axis, etc. See MoreOptions on the Dashboards builder for changing the show modality.                                                                                                                                                                                                                     |
|  | Multi Data Map (dashboard Map)                      | 1) Receive a JSON with a <b>set</b> of ServiceURI to be shown on map, eventual dynamic data pin, plus eventual controls for interacting with. You do not need to send data, only the ServiceURI, the data are automatically collected and updated by the Widget. You can dynamically change the ServiceURI.<br>2) The user acting on map, click on map is sending GPS position, and SURI (if clicked) on the input PIN of the node.<br>See for details <a href="https://www.snap4city.org/774">https://www.snap4city.org/774</a> |
|  | Event Driven MyKPI                                  | This node produces an event every time a given MyKPI changes its value.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|  | Synoptics                                           | This node can be used to read an indicated Synoptical Variable<br>See for details <a href="https://www.snap4city.org/644">https://www.snap4city.org/644</a>                                                                                                                                                                                                                                                                                                                                                                      |
|  | Synoptics                                           | This node can be used to send and event to an indicated Synoptical Variable<br>See for details <a href="https://www.snap4city.org/644">https://www.snap4city.org/644</a>                                                                                                                                                                                                                                                                                                                                                         |
|  | Synoptics                                           | This node produces an event every time a given Synoptical Variable changes its value.<br>See for details <a href="https://www.snap4city.org/644">https://www.snap4city.org/644</a>                                                                                                                                                                                                                                                                                                                                               |
|  | D3 charts                                           | A widget which may send to the dashboard complex graphic representation according to the D3 Library. The widget allows to provide to the Dashboard the specific D3 graph you like, the data and the interaction configuration you would like to have on it according to the D3 standard Library.<br>See for details <a href="https://www.snap4city.org/790">https://www.snap4city.org/790</a>                                                                                                                                    |

|  |                                    |                                                                                                                                                                                                                                                                                  |
|--|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | geolocator                         | The widget on dashboard to get the GPS location of the Browser of the client hosting the dashboard and send it to the flow from the output PIN of the flow.                                                                                                                      |
|  | Speak Synthesis                    | The flow can send a text which is vocalized on the client side, for example to help the user on browser, or to harm him. The voice can be of different languages, male/female, etc.<br>See for details <a href="https://www.snap4city.org/777">https://www.snap4city.org/777</a> |
|  | See Multi Data Map (dashboard Map) | DEPRECATED<br><a href="https://www.snap4city.org/774">https://www.snap4city.org/774</a>                                                                                                                                                                                          |
|  | See Multi Data Map (dashboard Map) | DEPRECATED                                                                                                                                                                                                                                                                       |
|  | See BarSeries                      | Almost DEPRECATED                                                                                                                                                                                                                                                                |
|  | See BarSeries                      | Almost DEPRECATED                                                                                                                                                                                                                                                                |
|  | See D3 charts                      | Almost DEPRECATED                                                                                                                                                                                                                                                                |

#### IV.C.4.c- Dashboard Widgets: rendering data/device tables on Dashboards

To implement some rendering of device/entity data on Table in some Dashboard/view there are multiple solutions according to what you need:

- A. **Table content widget** from Proc.Logic: it allows to who ServiceURI data you send and present some interaction. It can be also controlled from CSBL.
  - a. A simple and fast solution with some limitation
- B. **Device Table widget (server-side business logic)**: see above, it is highly configurable and accepts a list of ServiceURI plus some parameters and pass them from Proc.Logic Flow to show them on Dashboards. It may show configurable interactive buttons and tools.
  - a. The best solution for control room and prototypes.
- C. **External Content or Web Page Widgets (server-side business logic)**: prepare your own visualization as HTML/JavaScript (CSS) and sent it to it to widget/node to embed/show the HTML page. It may return to into the flow the data filled in forms and the interactivity performed on HTML page as well.
  - a. The best solution if you are producing some custom data table: more for Control Rooms.
- D. **Device Table widget (Client-Side business logic)**: see above, it is highly configurable and accepts a list of ServiceURI plus some parameters and are used to show them on Dashboards coding the data directly on Client-Side Business Logic in JavaScript as described in the following. It may show configurable interactive buttons and tools which can act on other widgets as well.
  - a. the best solution if you have many users....
- E. **External Content widget (Client-side business logic)**: prepare your own visualization as HTML/JavaScript and put it into the JavaScript field in the MoreOption of the widget. It may present forms, tables, and any element of HTML. It may interact and control all the other Widgets in the same Dashboards.
  - a. The best solution if you are producing some custom data table for large number of people.



#### IV.C.5. Development: Client-Side Business Logic

This approach is a prerogative of Snap4City development environment.

- See Client-Side Business Logic Widget

Manual: <https://www.snap4city.org/download/video/ClientSideBusinessLogic-WidgetManual.pdf>

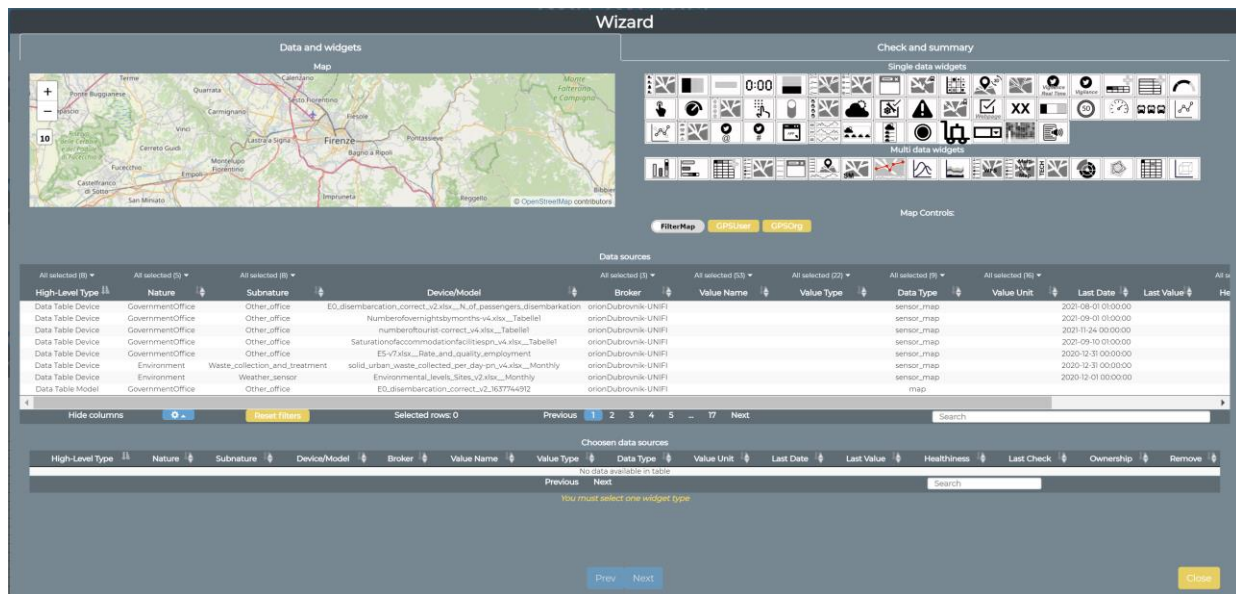
In Snap4City, Client-Side Business Logic is a solution to close the loop from user actions and effects on widgets directly on the client side, on the browser. This approach allows to have a **separated context for each user** since any Business Logic computation is performed on client side at every connection, thus reducing the computational workload on server side. Actually, **Client-Side Business Logic, CSBL**, and **Server-Side Business Logics, SSBL**, may be present at the same time behind a Dashboard and thus behind a Business Intelligence / Smart Application. This approach may be used to save some context and share with other users. In CSBL the logic code is formalized in JavaScript only, while in SSBL the logic is formalized in Proc.Logic which is Node-RED plus some JavaScript.

Thus, in CSBL we have IN, OUT and IN/OUT widgets (quite similar to SSBL):

- **IN Widgets** are those that are prepared to receive some actions/commands from the Users. For example, a click on a button, a click on the map, an action on a synoptic, etc. These IN Widgets can be regarded as Virtual Sensors, and may host JavaScript to specify what the Tools has to do when a certain User Action is performed.
- **OUT Widgets** are those that are prepared to provide some changes to be shown into the Users' interface. For example, a view of a barseries on some other data, a rendering of a time series, a rendering of a set of Entities on the map, etc. These OUT Widgets can be regarded as Virtual Actuators.
- **IN/OUT Widgets** are those that provide capabilities of both IN and OUT Widgets. For example, a map can receive an IN command about a selected PIN, and can receive an OUT command to show a selection of services, devices, etc. These IN/OUT Widgets can be regarded as Virtual Sensors/Actuators.

In the development of the **CSBL**, the JavaScript code is used, and it is activated by events / actions performed by the User on the IN Widgets. While the OUT Widgets are ready to receive commands from CSBL JavaScript, similarly to what they do when receive commands from SSBL via Web Socket.

The Developers that would like to develop CSBL have to be singularly authorized, please ask to [snap4city@disit.org](mailto:snap4city@disit.org). They will be entitled to go in the widget More Option and to find and edit the CK editor to formalize the JavaScript code according to a specific user manual provided.



**Figure 20: the Dashboard Wizard is used to create widgets in user Dashboards in the context of CSBL.**

When working in SSBL, widgets can be created and edited from Node-Red Processing Logic. When working in CSBL context, widgets can be created through the Dashboard Wizard, as shown in **Figure 20**.

The desired widget type can be chosen from the up-right section of the Wizard showing the available widgets. The desired data source can be chosen from the “Data Sources” table, exploiting the filters on column headers, as well as the text search box in the bottom-right corner of the same table. When the desired choices have been performed, by clicking on the “Next” button a final “Check and Summary” tab is shown for final check. At last, clicking on the “Create widget” magic wand the desired widget(s) are created in the current user dashboard.

In this way, the created widget(s) are in the form of OUT widgets, and they are ready to receive and show data and actions by JavaScript, as described in the table “**Commands which are ready to execute from JavaScript**” in the next pages.

In order to create IN widgets (those widgets which have this capability are listed in the following table “**Users’ Action Description and effects**”), an authorized developer can add the desired JavaScript code in the CK Editor box in the widget “More Options” box, as shown in **Figure 20b**. The JavaScript code should be provided as a single JavaScript function named “execute”. Please refer to the specific user manual for more detailed instructions.

# Modify widget

## Metric and widget choice

Widget category: Actuator  
Actuator target: Personal apps  
Input from personal apps: NR\_caa95069\_baa388  
Value type: Testuale  
Start value: {"options": "3382", "selected": ""}  
Domain type:  
Widget type: widgetImpulseButton

## Generic widget properties

Title: Trigger Pie C  
Background color: rgba(2  
Content font size:  
Header color: rgba(5  
Header text color: rgba(2  
Period:  
Refresh rate (s):  
Height: 10  
Width: 11  
U/M:  
U/M position:  
Show header: Yes  
Font type (autosuggestion): Auto

## Specific widget properties

View mode: Icon and text  
Impulse mode:  
Button color: rgba(214,2  
Button color on click: rgba(214,2  
Symbol color: rgba(0,0,0  
Symbol color on click: rgba(0,0,0  
Text color: rgba(0,0,0  
Text color on click: rgba(0,0,0  
Text font size: 24  
Display font size: 24  
Display text color: rgba(255,2  
Display text color on click: rgba(255,2  
Display background color: rgba(0,0,0  
Display radius (%):  
Display width (%):  
Display height (%):

Enable CK Editor: yes

Here you can insert Javascript code to be executed in the widget. Please save your script by clicking on the save button on the bottom.

```
function execute() {
 $('body').trigger({
 type:
 "showPieChartFromExternalContent_w_AggregationSeries_2573_wi
 dgetPieChart34123",
 eventGenerator: $(this),
 targetWidget:
 "w_AggregationSeries_2573_widgetPieChart34123",
 color: "#e8a023",
 color2: "#9c6b17",
 widgetTitle: "Vehicle Flow from Impulse Button",
 });
}
```

**Figure 20b. Detail of the CK editor (highlighted in red) in the widget “More Options” box.**

IN and IN/OUT Widgets which present the possibility of scripting in JavaScript when an action is performed on their graphic user interface are reported in the following table. The performed action by the user provoke the activation of a call back that can be filled in the JavaScript editor of the Widget to formalize the action to be performed. At the action, a number of parameters can be provided. For example, at a click on map the coordinates, etc. Into the JavaScript, the developer can code how this information can be used to command OUT Widgets. Data can also be autonomously recovered by an OUT widget using REST Calls to Smart City API and other activities. Please note that OUT Widget are typically capable of making queries to the Smart City API just sending them the SURJ of the Entities to be recovered, etc. This let the developer define a sort of **general-purpose widget** (i.e., not linked to a specific data source/device) that can retrieve data using API with specific input given at run-time. For example, suppose to have a table widget (IN widget) listing devices and a single pie-chart widget (OUT widget) on a dashboard. After a user’s click on a listed device, the table widget sends the device ServiceURI to the pie-chart widget that can update itself by calling an API to get the data, and show the new information related to the clicked device.

### See CSBL manual for an updated table

| IN and IN/OUT Widgets       | Users' Action Description and effects                                                                                                                                                                                 |                 |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| widgetTimeTrend             | Drill-Down on time interval selection (zoom), providing, SURI, value name, start and end time stamp                                                                                                                   | Time            |
|                             | Send Reset Drill-Down                                                                                                                                                                                                 | Time            |
|                             | Click on a single time instant, providing time stamp, SURI and value name                                                                                                                                             | Time Selection  |
| widgetMap<br>(multidatamap) | Click on a generic point on the map, providing coordinates (under development, currently it only works for SSBL)                                                                                                      | Space Selection |
|                             | Click on a PIN, providing coordinates and ServiceURI of the clicked PIN                                                                                                                                               | SURI Selection  |
|                             | Select the bounding box area shown on the map, and the zoom level in order to perform geographical Drill-Down on the entities (devices identified by SURIs, Points of Interest etc.) which are currently shown on map | Space Selection |
| widgetPieChart              | Click on a sector that identifies the name of a metric, providing: value, timestamp, entity name (from which the SURI can be reconstructed) value name, value type and value unit                                     | SURI Selection  |
|                             | click on a sector that identifies a device ID or MyKPI ID, providing: value, timestamp, entity name (from which the SURI can be reconstructed) value name, value type and value unit                                  | SURI Selection  |
|                             | Get date and time from the calendar (under development)                                                                                                                                                               | Time Selection  |
|                             | Click on legend, providing the status (e.g.: "checked" or "unchecked") of the metric/SURI which has been clicked (under development)                                                                                  | SURI Selection  |
| widgetBarSeries             | Click on a bar, providing: value, timestamp, entity name (from which the SURI can be reconstructed) value name, value type and value unit                                                                             | SURI Selection  |
|                             | Get date and time from the calendar (under development)                                                                                                                                                               | Time Selection  |
|                             | Click on legend, providing the visibility status of each metric/SURI                                                                                                                                                  | SURI Selection  |
| widgetRadarSeries           | Click on a radar axis related to a specific metric of a specific device, providing: value, timestamp, entity name (from which the SURI can be reconstructed) value name, value type and value unit                    | SURI Selection  |
|                             | Get date and time from the calendar (under development)                                                                                                                                                               | Time Selection  |
|                             | Click on legend, providing the visibility status of each metric/SURI                                                                                                                                                  | SURI Selection  |

|                                          |                                                                                                                                                                                                     |                      |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| widgetCurvedLineSeries<br>(multi series) | Drill-Down on time interval selection (zoom), providing: start and end time stamp, and list of SURI. It is also possible to program the synchronization of multiple widgetCurvedLineSeries widgets. | Time                 |
|                                          | Click on a single time instant, providing: time stamp and list of objects including SURIs and related entity names and value names                                                                  | Time Selection       |
|                                          | Click on legend, providing the visibility status of each metric/SURI                                                                                                                                | SURI Selection       |
|                                          | Get date and time from the calendar (under development)                                                                                                                                             | Time Selection       |
|                                          | Send Reset Drill-Down                                                                                                                                                                               | Time                 |
| widgetDeviceTable                        | Click on the action buttons, providing the action type, the corresponding SURI and a list of attributes with their corresponding values                                                             | SURI Action          |
| widgetImpulseButton                      | Click on button as a trigger (no parameters are provided)                                                                                                                                           | Action               |
| widgetOnOffButton                        | Click on button, providing the new status                                                                                                                                                           | Action               |
| widgetKnob                               | Drag on knob, providing the value selected on the knob                                                                                                                                              | Action               |
|                                          | Click on minus and plus action (under development)                                                                                                                                                  | Action               |
| widgetNumericKeyboard                    | Click on the confirm button, providing the numeric value typed on the keyboard                                                                                                                      | Action               |
| widgetEventTable                         | Click on the action buttons, providing the action type, the corresponding event SURI and the ordering criteria                                                                                      | SURI Action          |
| widgetExternalContent                    | It can support HTML pages and SVG Synoptics, in addition to JavaScript, so that it can perform a wide range of actions that can be defined in the HTML/SVG/JS code by the users.                    | Depending on program |

OUT and IN/OUT Widgets which present the possibility of receiving commands from JavaScript of IN Widgets are reported in the following table. Each OUT Widget to be controlled has to be identified. The identifier of each Widget is accessible from the More Options of the widget, the so-called Widget name.

In most cases, the command to be sent to OUT Widget include a JSON very similar or identical to the JSON to be send to the widget from the Node-RED in the context of writing CSBL. So that, please refer to the same help manual and web pages for the same widgets.



### See CSBL manual for an updated table

| OUT and IN/OUT Widgets | Commands which are ready to be executed from Widget according to JavaScript in some IN Widget                                                                                                                                                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| widgetPieChart         | <p>Receive a JSON containing a list of SURI, metric names and/or values, and show their corresponding values on a Pie Chart graph.</p> <p>set date and time from the calendar (under development)</p>                                                                                                                              |
| widgetRadarSeries      | <p>Receive a JSON containing a list of SURI, metric names and/or values, and show their corresponding values on a Radar/Kiviat graph.</p> <p>set date and time from the calendar (under development)</p>                                                                                                                           |
| widgetBarSeries        | <p>Receive a JSON object containing a list of SURI, metric names and/or values, and show their corresponding values on a Bar graph.</p> <p>set date and time from the calendar (under development)</p>                                                                                                                             |
| widgetSingleContent    | <ul style="list-style-type: none"> <li>Receive a SURI and a metric name, or a value, or a text string, and show the corresponding value.</li> <li>Receive and show a HTML/JS page</li> </ul>                                                                                                                                       |
| widgetSpeedometer      | Receive a SURI and a metric name, or a value, and show the corresponding value on a speedometer graph.                                                                                                                                                                                                                             |
| widgetGaugeChart       | Receive a SURI and a metric name, or a value, and show the corresponding value on a gauge graph.                                                                                                                                                                                                                                   |
| widgetTimeTrend        | <p>Receive a SURI and a metric name, or a value, and show the corresponding time-series on a line, spline, area or stacked area graph.</p> <p>Receive reset zoom</p>                                                                                                                                                               |
| widgetTable            | <p>Receive a JSON containing a list SURI, metric names and/or values, and show the corresponding time-series on a HTML static table.</p> <p>Receive start datetime, end datetime without change sources IDs (under development)</p>                                                                                                |
| widgetCurvedLineSeries | <p>Receive a JSON containing a list of SURI, metric names and/or values, and show the corresponding time-series on a line, spline, area or stacked area graph.</p> <p>Receive start datetime, end datetime without change sources IDs</p> <p>set date and time from the calendar (under development)</p> <p>Receive reset zoom</p> |
| widgetDeviceTable      | Receive a JSON containing a list of SURI representing IoT devices, and show their related attributes and values on an interactive table which provides action buttons.                                                                                                                                                             |
| widgetEvent            | Receive a JSON containing a list of SURI representing events as virtual devices, and show their related attributes (e.g., start and end date) and values on an interactive table which provides action buttons.                                                                                                                    |
| widgetMap              | Receive a JSON containing a list of SURI or entities (such as heatmaps, categories of Points of Interest etc.) and show them on an interactive map as clickable markers, dynamic SVG pins, traffic flows, heatmaps etc.                                                                                                            |
| widgetOnOffButton      | Receive and show a value representing the status                                                                                                                                                                                                                                                                                   |
| widgetKnob             | <p>Receive and show a value</p> <p>set date and time from the calendar (under development)</p>                                                                                                                                                                                                                                     |
| widgetNumericKeyboard  | Receive and show a value                                                                                                                                                                                                                                                                                                           |

Other functions which can be exploited on Actions JavaScript segments:

- **Open a New Dashboard:** `openNewDashboard ( )`
- **Get parameters:** `getParams ( )`

In this manner, using the `openNewDashboard()` function after receiving an event (e.g., an user's click), it is possible to open a new dashboard and sent to it some parameters. Widgets in the new dashboard can then implement the `getParams()` function to retrieve the sent data. This approach gives the possibility to implement a kind of *template dashboard*, where several widgets are included but not linked to specific devices. When the template dashboard is called (with the `openNewDashboard()` function) its widget can be populated with data (obtained with the `getParams()` function). The passed data can include a full JSON, such as those used to configure the widget in Node-RED, or simply a list of ServiceURIs of some devices of interest, in which case the widget will use the Snap4City API to get the device measurements.

#### IV.C.6. Development: external applications and Data Analytics using the Advanced Smart City API

The external APIs of Snap4City are documented in Swagger:

<https://www.km4city.org/swagger/external/index.html>

**Advanced Smart City APIs are:**

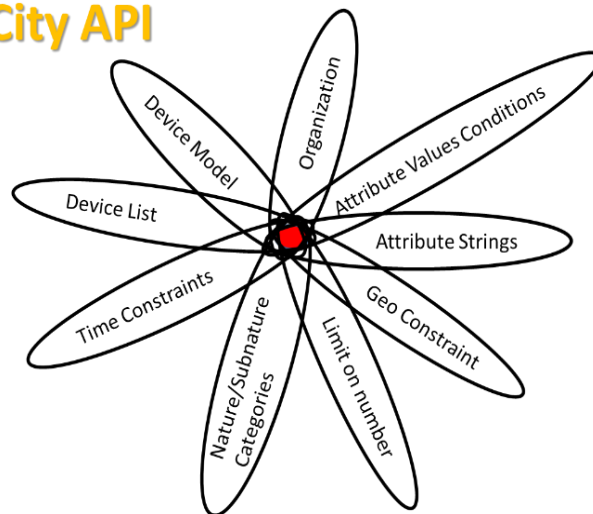
- Smart City API
  - **Mainly to get data from the platform**
  - To access the KB, Service Map resources and query
  - Several filtering and capabilities, see also section on Dashboards.
  - See this section.
- Orion Broker API, NGSI V2
  - **Mainly to send/store data/messages on platform** via registered entities
  - To communicate with Orion Brokers exploiting the Secure Filter of Snap4City.
  - **See Section IV.C.9**
- Entity Directory (IoT Directory) API
  - To register new devices, pose queries, etc.
- MyKPI API
  - To access (read data, and send data) at MyKPI and other personal data safer
- Heatmap, etc., API
  - To save and access to HeatMaps of the Heatmap server.
  - Similarly, it can access to OD matrices, Traffic Flows, etc.

**Authentication mechanisms for all of them are reported in Section IV.C.8.**

**All the Advanced Smart City API may be managed by an API Management system based on APIMAN for accounting and billing if needed to control business, traffic, usage, impose limitations, etc. See: <https://www.snap4city.org/827>** APIMan management tool can be placed in front of the several APIs to (i) provide a unified entry point, (ii) perform accounting on the API consumption, (iii) placing the basis for providing billing on the basis of API usage and data consumption according to different business models: (a) monthly rate, (b) limits, (c) pay per play, etc. **APIMAN** may have an integrated SSO with the rest of the Snap4City platform or an independent KeyCloak or other Authentication service connected.

## Selection on Smart City API

- Combining different filters for selecting entities from Smart City APIs
- Be care:** filtering too much may lead to empty set 😊



**Figure 21: Smart City API filtering capabilities.**

Smart City APIs provide a large number of filtering capabilities as described in training course part 2, and part 4<sup>th</sup> and summarized in Figure 21.

Each instance of the Snap4City Platform has its own Smart City API, in addition, there is an integrated federated service (called Super) to query all the installations which are federated each other (the federation is decided by each platform owner). Several separated federations may be created as well. Each installation may expose both local Smart City API and Super Smart City API. They have the same syntax but different prefix call.

In each call to the API, you can

- filter by category (nature, Subnature), etc., to obtain a set of devices data and thus also their ServiceURI.
- Retrieve the data of the single Entity Instance (device)/ServiceURI.
- Retrieve the data of a set of Entity Instances (devices)/ServiceURIs.

For example, on using Smart City APIs: (see training slide part 8<sup>th</sup>)

- Please note that:
  - each installation of Snap4City platform may have its own ServiceMap and SUPER to which one can refer. Please ask to your platform administrator.
- via regular Smart city API by category, etc.
  - [http://svealand.snap4city.org/ServiceMap/api/v1/?selection=59.581458578537955;16.71183586120606;59.62875017053684;16.875171661376957&categories=Street\\_light&maxResults=100&format=json](http://svealand.snap4city.org/ServiceMap/api/v1/?selection=59.581458578537955;16.71183586120606;59.62875017053684;16.875171661376957&categories=Street_light&maxResults=100&format=json)
  - This case is valid for **Svealand** organization and KB direct access to smart city API
- Via Super
  - <https://www.disit.org/superservicemap/api/v1/?.....>
  - This case is valid for **SUPER on snap4city.org**.
- Via Super by values
  - <https://www.snap4city.org/superservicemap/api/v1/iot-search/?selection=43.77;11.2&maxDists=700.2&model=CarPark>
  - <https://www.snap4city.org/superservicemap/api/v1/iot-search/?selection=42.014990;10.217347;43.7768;11.2515&model=metrotrafficsensor&valueFilters=vehicleFlow>0.5;vehicleFlow<300>

- This case is valid for **SUPER on snap4city.org**.

Time Series are attached to Devices which are identified by ServiceURI. To access at the Time Series (also called real time data) you can:

- From Processing Logic (IoT App) use the block «service info dev». **In this case, you automatically access to your private and delegated data. You do not need to perform the authentication since it is performed directly from the microservice Processing Logic (IoT App) context, both on cloud and on edge.**
- From Python/Rstudio, Web and Mobile App, you can call Smart City API, see in this section and in Part 7 of the course.
- (1) Retrieve data from Processing Logic (IoT App) and pass them to Python/Rstudio in JSON as presented in other sections, or (2) pass to them only some parameters such as the GPS location and categories. This approach is viable for small amount of data, such as some thousands. For larger amount of data or to be more efficient we suggest using case (2) which is a direct access to the Smart City API from the Data Analytic.

**Access to public data does not need to be authenticated, the Smart City API can provide public data without any authentication.**

**Access to private data from Smart City API can be performed according to the Open ID connect approach. You have to get the Access Token and then perform your API call for query selection as above.**

**In the following Sections and in the training course sections you can get an example about how to get access to the Private data, and to send data on the platform in the secure manner:**

<https://www.snap4city.org/download/video/course/da/>

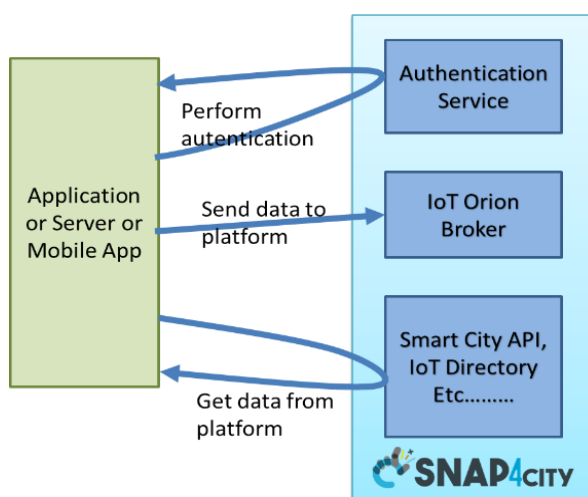
<https://www.snap4city.org/download/video/course/app/>

For each Organization on Snap4City.or and for each Organization on any Snap4City installation may provide different: brokers and KB. In addition, for the authentication using OpenID connect as explained in the next section, the so-called client-ID is need. Therefore, please ask to the administrator for your endpoints and clientid, for example:

- **Orion broker NGSI V2 url:** <https://iot-app.snap4city.org/orion-broker-xxxxxx/v2/>
- **API base url:** <https://yyyyy.snap4city.org/ServiceMap/api/v1/>
- **Client id for keycloak:** zzzzzzzidzzzz (the client is public, no client secret is needed)

#### IV.C.6.a- Authentication to API access, REST Call

The approaches for integrating with the platform are reported in **Figure 22**.



- **First**
  - authenticate
- **Second**
  - use the services to provide and/or get data

**Figure 22: Authentication with Snap4City platform.**

The Approach adopted is based on OpenAuthentication, Open ID Connect, using KeyCloak tool. For the Authorization we use LDAP for the users' roles and the Authorization Manager. OAuth2.0 one of the most used authentication protocols for web applications, standardized by IETF, used by many big social networks (Facebook, google, twitter, github, ...)

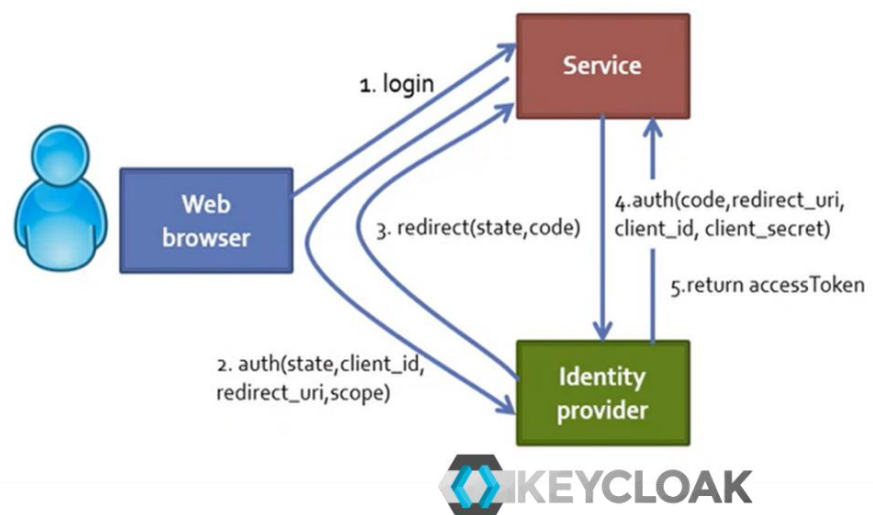
#### Different cases for OAuth 2.0:

- **web server application**
  - browser client and some application running on server side (service)
- **Single page App**
  - application running on the browser, no specific server-side code.
- **Implicit Flow Protocol, mobile application**
  - application running on mobile, no specific server-side code.
  - more secure than a stand-alone web page since some secret may be stored into the mobile app.

Please note that both client\_id and client\_secret have to be requested to Snap4City platform administrator. If client\_secret should be only used by server-side server which can keep it in some safe. All the Access Token are in JWT, see them on <https://www.jwt.io>

#### Web Server Application:

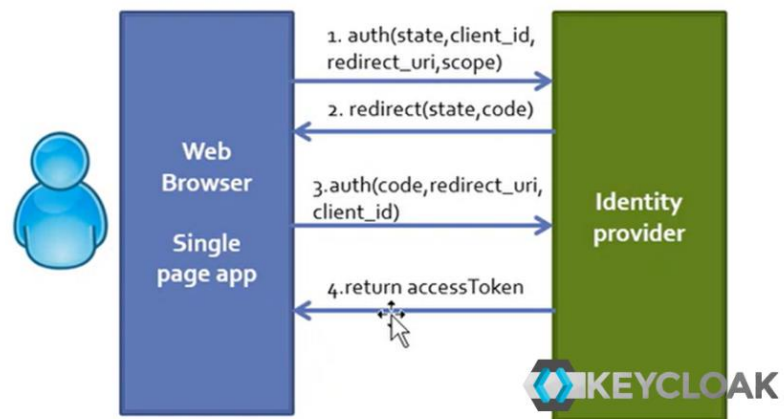
- web browser opens the url, the HTML pages are produced by the Service, which is a Web Application, the logic is on Server Side, the service redirect on the Identify Provider.
- The Service generates a random state string (saved in the session) and redirects to Identify provider (1)
- (2) The Identify Provider checks client\_id and redirect\_uri
- user can login (if not already done), user is requested permission for the Service and if authentication succeeds and user give permission the Identify Provider redirects (3) to redirect\_uri?code=...&state=... of the Service
  - code is used to get an access token.
  - state is used to check that the call was performed by this site, an ID generated by the Service.
  - the Service receives the code and state (3) back, checks the state is the same as the one originally sent (2).
- the Service uses the «code» to get an access token, making (4) a request to the Identify Provider in Post (this time passing client\_id and client\_secret, etc.). Finally, the Service receives the Access Token (5) from the Identify provider.
- The Service can now make a request to the API to get information of the users' data or send some data.



**Single page App on some server, without web application server logic.**



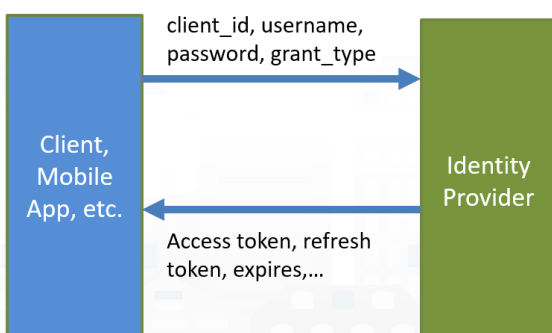
- Only browser without the server-side app, or better, the service side app role is played by the Identity Provider.
- In this case the Single Page App is running on the web browser, the client\_secret cannot be used to get the access token.
- web browser opens the url, the HTML pages are produced bringing the authentication on Identity Provider.
- (1) The authentication of the client is performed by using the client\_id, state, redirect\_uri to be called and scope.
- (2) The Identity provider redirect to the single page app since the single page is accessible as a web page.
- (3) finally, the web page can use the received code for getting the accessToken (4).
- Then the web page can make a request to the API to get or send information to the portal.



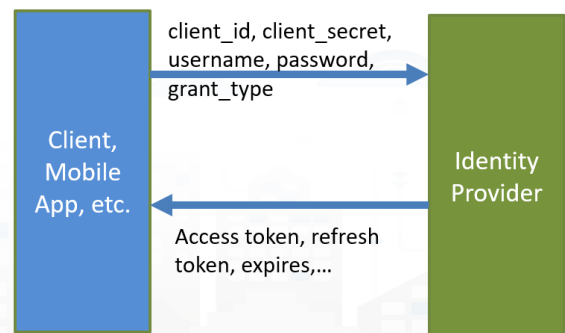
#### Implicit Flow Protocol: public applications, mobile application, (data analytics in Python and RStudio)

- Less secure: It's not recommended to use this flow unless you absolutely need to
  - For Front-End Web Application that do not have Server-Side functionalities.
  - It may be used from cloud data analytics processes, for example.
  - If you develop a client application in JavaScript, you can only go with this approach.
- These clients have to use the client\_id and may not use the client\_secret depending on the kind of client (if application of the client\_id is public the secret is not needed)
  - The client\_secret has to be obtained and saved only by server applications. For example, in the case above of Web Server Application above.
- The Mobile App use the Client\_id, username and password to directly access getting the Access Token and the Refresh Token.
- Then the Mobile App can now make a request to the API to get information of the users' data or send some data.

#### Public Applications



#### Implicit Flow Protocol



Therefore, the Implicit Flow Protocol is the most secure and suitable for the mobile apps, the client secret can be omitted if needed.

The solutions are described in the slides of part 8 of development:

<https://www.snap4city.org/download/video/course/dev/>

#### IV.C.6.b- Example: Access to Orion Broker API

**Send data message on Broker: send values of one or more attributes of a device sending data to Orion Broker: please use the call for UPDATE values, POST is not needed.**

Please note that at the first registration of a Device /Entity the full set of variables is provided and thus any successive data may arrive by using the PATCH, making POST not needed. If you make some changes on the Device/Entity, the system is also providing the changes into the Storage, and thus after the Device update in structure you can continue to use the PATCH.

Please refer/replace to:

- SWAGGER for external API: <https://www.km4city.org/swagger/external/index.html>
- your domain KB: mentioned in the example has <https://<xxxxx>>
  - you KB of reference can be recovered from the left side menu
- your domain broker mentioned in the examples: orion-broker-XXX
  - the broker name can be recovered from IoT Directory
- your deviceid mentioned in the example: <deviceId>
  - the device id can be recovered from IoT Directory, as the SURI
- your service URI, SURI mentioned in the example: Service URI
  - can be recovered from IoT Directory, ServiceMap, etc.

**This example refers to NGSI V2 ORION Broker**

PATCH [https://<xxxxx>/orion-broker-XXXX/v2/entities/John\\_Doe\\_Question/attrs?type=Sensor&elementid=John\\_Doe\\_Question](https://<xxxxx>/orion-broker-XXXX/v2/entities/John_Doe_Question/attrs?type=Sensor&elementid=John_Doe_Question)

Authorization: Bearer <accesstoken>

```
{
 "dateObserved": {
 "value": "2023-01-14T01:01:02.000Z",
 "type": "string"
 },
 "questionnaire": {
 "value": "Q1",
 "type": "string"
 }
}
```

The accesstoken has to be of the device owner or of a user with read-write delegation.

**GET last value from Orion Broker (it should be carefully used since the broker does not provide data with the ServiceURI, and does not provide historical data, please use Smart City API)**

GET <https://<xxxxx>/orion-broker-XXXX/v2/entities/<deviceid>?type=<devicetype>&elementid=<deviceid>&k1=<k1>&k2=<k2>>

deviceid, devicetype, k1 and k2 can be retrieved from IoT Directory, K1 and k2 are two keys allowing read/write access to device. For example:

[https://<xxxxx>/orion-broker-XXXX/v2/entities/John\\_Doe\\_Question?type=Sensor&elementid=John\\_Doe\\_Question&k1=<k1>&k2=<k2>](https://<xxxxx>/orion-broker-XXXX/v2/entities/John_Doe_Question?type=Sensor&elementid=John_Doe_Question&k1=<k1>&k2=<k2>)

use of k1 & k2 for authentication is very simple, but it deprecated (give complete access to the device).

Using access token is more secure:

GET [https://<xxxx>/orion-broker-XXXX/v2/entities/John\\_Doe\\_Question?type=Sensor&elementid=John\\_Doe\\_Question](https://<xxxx>/orion-broker-XXXX/v2/entities/John_Doe_Question?type=Sensor&elementid=John_Doe_Question)

Authorization: Bearer <accesstoken>

The access token should be of the owner of the device or of a user that has read only or read/write delegation.

#### IV.C.6.c- Example: forbidden characters by the Orion Broker

In order to avoid script injections attack in some circumstances (e.g. cross domain to co-located web servers in the same hot that CB) the following characters are forbidden in any request:

< > " ' = ; ( )

URL encoding is a valid way of encoding. However, we don't recommend its usage for fields that may appear in API URL (such as entity id or attribute names) due to it would need to encode the "%" character itself. For instance, if we would want to use "E<01>" as entity id, its URL encode would be: "E%3C01%3E".

In order to use this entity ID in URL (e.g., a retrieve entity info operation) the following will be used (note that "%25" is the encoding for "%").

See more on [https://fiware-orion.readthedocs.io/en/2.4.0/user/forbidden\\_characters/index.html](https://fiware-orion.readthedocs.io/en/2.4.0/user/forbidden_characters/index.html)

#### IV.C.6.d- Example: Get Data using Smart City API

##### **Direct Data Access from storage/KB without using the Super of federated access**

GET <https://<xxxx>/ServiceMap/v1/?serviceUri=...&fromTime=...&toTime=...>

Authorization: Bearer <accesstoken>

Where the service uri is

<http://www.disit.org/km4city/resource/iot/orionxxxxxx/yyyyyyyyyy/<device id>>

see <https://www.km4city.org/swagger/external/> for more details.

#### IV.C.6.e- Example of Registering a New User via Mobile App

POST <https://www.snap4city.org/drupal/api/user/register>

ContentType: application/json

Accept: application/json

```
{
 "name": "John.Doe",
 "mail": "jdoe@me.com",
 "legal_accept": "true",
 "extras-1": "true",
 "extras-2": "true",
 "og_user_node": { "und": [{ "default": "722" }] }
}
```

// note that:

722 = Organization is "Greece-UNISYSTEMS"

319 = Organization is "DISIT"

Etc.

**Return 200** if user can be registered, an email is sent to the user to finish registration on the server (set password) the result is:

```
{ "uid": "8005",
 "uri": "https://www.snap4city.org/drupal/api/user/8005"
}
```

**Return 406** in case of error like existing username or existing email address.

```
{ "form_errors": {
 "name": "The name <em class=\"placeholder\">p.bellini is already taken."
 }
}
```

This approach can be used to register the new user from a mobile or Web App. In any case, the finalization of the registration will have to be performed (on the basis of the links received on the email address provided) on the Portal (in general case on snap4city.org, or other snap4tech platform from which the registration is requested). This two steps registration is a minimal secure practice.

*It has no sense to provide the possibility of setting a password in one shot via an API only by knowing a user ID, since it would allow to any user to change the password of any other users.*

Once the registration is finalized on the portal providing a preferred password, the user can enter in the mobile App with the credentials, and the mobile App can use the following functionalities to get Access Token to send data messages on broker via NGSI V1/V2, as well as to get data from smart city API.

#### IV.C.6.f- Example: Get access token/refresh token via user credentials

To get access token and refresh token associated with a user the following REST API is available:

**POST** <https://www.snap4city.org/auth/realms/master/protocol/openid-connect/token>

Content-Type: application/x-www-form-urlencoded

grant\_type=password&client\_id=xxx&username=...user...&password=...password...

This is the case where the client\_id xxx is a public client (a mobile app or a one page app), if it is a confidential client also the client\_secret parameter needs to be provided. The clients need to be registered in keycloak.

**Return 200** if the user and password match

```
{
 "access_token": "...",
 "expires_in": 1500,
 "refresh_expires_in": 2073600,
 "refresh_token": "...",
 "token_type": "bearer",
 "not-before-policy": 0,
 "session_state": "..."
}
```

The access\_token has a limited lifetime (1500 seconds) after that period you can use the refresh token to request a new access token, also the refresh token has a limited life time.

**Return 401** if the user credentials are not correct or the client\_id is not present

```
{
 "error": "invalid_grant",
}
```

```
"error_description": "Invalid user credentials"
}
```

#### IV.C.6.g- Example: Get access token via refresh token

To get access token using the refresh token acquired during authentication the following REST API is available:

**POST** <https://www.snap4city.org/auth/realms/master/protocol/openid-connect/token>

Content-Type: application/x-www-form-urlencoded

```
grant_type=refresh_token&client_id=xxx&refresh_token=...refresh token...
```

**Return 200** if the refresh token is valid, access token is returned

```
{
 "access_token": "...",
 "expires_in": 1500,
 "refresh_expires_in": 2073600,
 "refresh_token": "...",
 "token_type": "bearer",
 "not-before-policy": 0,
 "session_state": "..."
}
```

**Return 400** if the refresh token is not valid or the client\_id is not present

```
{
 "error": "invalid_grant",
 "error_description": "Invalid refresh token"
}
```

**Note:** each authentication request creates a new session on keycloak, it is better to use the refresh token to create a new access token, it is more secure (no need to store user name and password)

#### IV.C.6.h- Example: Using Advanced Smart City API in Python

See example of the usage of Python for developing MicroService and make it accessible from Proc.Logic /IoT app: <https://www.snap4city.org/645>

See also Python usage of API: <https://www.snap4city.org/829>

### get token in Python

```
import requests
```

```
this is only an example do not use in production,
consider that each call it will create a new session on keycloak for each request,
the token should be reused until it is expired and when expired it can be 'refreshed' using the refresh token
```

```
def getTokenViaUserCredentials(username,password):
 payload = {
 'f': 'json',
 'client_id': 'xxxxx-tool',
 'grant_type': 'password',
 'username': username,
```



```
'password': password
}

header = {
 'Content-Type': 'application/x-www-form-urlencoded'
}

urlToken = "https://www.snap4city.org/auth/realms/master/protocol/openid-connect/token"
response = requests.request("POST", urlToken, data=payload, headers=header)
token = response.json()
return token

def getTokenViaRefreshToken(old_token):
 if not 'refresh_token' in old_token:
 return None

 payload = {
 'f': 'json',
 'client_id': 'xxxxx-tool',
 'grant_type': 'refresh_token',
 'refresh_token': old_token['refresh_token']
 }

 header = {
 'Content-Type': 'application/x-www-form-urlencoded'
 }

 urlToken = "https://www.snap4city.org/auth/realms/master/protocol/openid-connect/token"
 response = requests.request("POST", urlToken, data=payload, headers=header)
 token = response.json()
 return token
```

### Access data via Smart City API: get historical and last data via ServiceURI, in Python

```
#get the token of the user
token = getTokenViaUserCredentials ('...user...', '...password...')
print('token', token)
if 'access_token' in token :
 head = {
 "Content-Type": "application/json",
 "Authorization": f"Bearer {token['access_token']}"
 }
 api_url = "https://<xxxxx>/ServiceMap/api/v1/"
 service_uri = "http://www.disit.org/km4city/resource/iot/<xxxxx>/<xxxxx>/cccc Device_01"
 response = requests.request("GET", api_url + "?serviceUri=" + service_uri + "&fromTime=60-day",
 headers=head)
 print(response.json())
```

else:

```
print('failed getting access token', token)
```

### Send data on platform, via Broker: send/update values of a device, in Python

```
if 'access_token' in token :
 head = {
 "Content-Type": "application/json",
 "Authorization": f"Bearer {token['access_token']}"
 }
 api_url = "https://iot-app.snap4city.org/orion-broker-<xxxxx>/v2/"
 device_id = "cccc_Device_01"
 payload = {
 "dateObserved": {
 "value": "2023-01-27T01:01:02.000Z",
 "type": "string"
 },
 "FatigueValue": {
 "value": 13.1,
 "type": "number"
 }
 }
 response = requests.request("PATCH", api_url + "entities/" + device_id
 + "/attrs?type=Sensor&elementid=" + device_id, json=payload, headers=head)
 if response.status_code!=204 :
 print(response.json())
 else:
 print(device_id+" updated!")
else:
 print('failed getting access token', token)
```

#### IV.C.6.i- Snap4City Internal API, REST Calls

Typically, the Internal API can be used from modules and tools which are located on the intranet of the platform installation. <https://www.km4city.org/swagger/internal/index.html>

In details they are:

- **IoT Directory API: IoT Devices and tools API:**
  - **IoT Device registration API:** API of the Entity / IoT Directory.
  - **Sensors API:** API of the Entity / IoT Directory.
  - **Device, Broker, and Value Mgmt API:** API of the Entity / IoT Directory.
- **Mobile App Management**
  - **User Profiler API:** To manage the user profile for the **Engager** on Mobile Apps.
  - **Engager API:** From the Engager to prepare engagements to the Mobile Apps.
  - **Wallet API:** From the Engager to Wallet o the users of Mobile Apps and in general.
  - **Snap4City Application API.**
- **Resources and entities Management**
  - **Snap4City Application API:** To manage Proc.Logic / IoT Apps.
  - **My KPI API:** To manage MyKPI, MyPOI, POI, etc.

- **Data Manager API:** to access personal data.
- **Resource Manager API:** To manage resources in the marketplace.
- **Ownership API:** To manage ownerships and delegations.
- **Device Groups API:** To manage ownerships and delegations.
- **Event Logger API:** to log data.
- **Snap vs OpenMAINT API:** Integration with the workflow management, BPM, and ticketing.

## IV.D. Testing and Deploy Phases

It is the mandatory phase in which the solutions (with IoT Apps and/or Dashboard) is deployed on Cloud or on IoT Edge. Only after the deploy the Solution is put in execution and can be tested. It is the phase in which all the needed testing are performed. Unit testing, regression testing, etc. The test is different for the different development parts. The IoT App testing is performed once deployed the IoT App, the flow is saved, and the IoT App can be tested using Inject and Debug functions, but also controlling the effects to the external services of the IoT App via corresponding microservices.

The activities of Test and Deploy are performed into the corresponding tools

- **IoT App Editor Node-RED** provides a button for Deploy and a Debug console for testing
- **Data Analytics** are
  - tested on development user interface on RStudio and Python
  - Tested on Deploy when they are executed as container from IoT Apps
- **Dashboards** are tested directly into the Dashboard editor and preview.

Please note that the Snap4City development environment allows to have on the same environment applications and solutions which are both in phase of development and finalized.

On the same infrastructure, production tool and testing tools may be executed and used at the same time. The infrastructure is robust enough to do not crash to the fault of your testing solutions if they are not aggressively destroying the running processes.

In alternative you can use provided Snap4City Development environment which can be installed in a single VM or via Docker compose.

- <https://www.snap4city.org/738>

## IV.E. Validation and Production Phases

It is the final phase, which is reached after the acceptance testing, thus the specific smart solution is put in production and made accessible for the users.

Validation is the verification of the solution wrt complete use cases, the process should be performed by different personnel with respect to the developers which also performed the testing and deploy. For this phase actual data is used and actual connector and user interfaces. It is the final phase, which is reached after the acceptance testing, thus the specific smart solution is put in production and made accessible for the users.

Is the phase in which all components can be integrated and tested in their integration on the platform ready to be used in production.

The **validation** should be performed verifying:

- Functional Requirements
- Nonfunctional Requirements

The **production** process is very easy in Snap4City since implies to provide access to the tools and services to final users you planned. The grant can be performed on Dashboard Management and on IoT Directory, and on Data Management for the data. Once the solution is put in production it can be monitored in deep on



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

**DINFO**  
DIPARTIMENTO DI  
INGEGNERIA  
DELL'INFORMAZIONE

**DISIT**  
DISTRIBUTED SYSTEMS  
AND INTERNET  
TECHNOLOGIES LAB



Dashboard usage, on data status, on IoT App, etc. See Part 6 of the training course.

## V. Other Aspects

### V.A. Authentication and Authorization, integration

- **Authentication in Snap4Tech is based on KeyCloak which is based on SAML,** <https://auth0.com/blog/how-saml-authentication-works/>
- **Different Versions of interoperability Authentication and Single Sign On, SSO,** are available on demand, with
  - **Spid**, Public Digital Identity System, <https://www.spid.gov.it/en/>
  - **EIDAS** (electronic IDentification Authentication and Signature), <http://www.agid.gov.it/en/platforms/eidas>, <https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation>
  - **CIE**, Electronic Identity Card [https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity\\_en](https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en)
  - **RealMe NZ**, <https://www.realme.govt.nz/>

### V.B. Creation of Smart Applications exploiting the platform

**Smart Applications** exploiting the platform:

- The above process allows to create NATIVE smart applications provide from server.
- External web and Mobile applications can be developed by using the Advanced Smart City API of Snap4City
- Third Party Applications and Solutions can be integrated into the solution in different manners.

**Creation of third-party Web and Mobile Applications exploiting the platform:**

- **Advanced Smart City APIs and MicroServices** and modalities to create traditional and advanced Web and Mobile Applications and services, all on Node-RED platform on IoT Edge and on Cloud;
- **Applications (can be Native or Third party):** smart parking, smart biking, smart light, control systems, control room, energy management, smart bed, smart manufacturing, smart building, mobility and transport, security, etc.

### V.C. Nonfunctional aspects of the Snap4City Development Environment

- **Online development tools**, from data modelling to data ingestion, from data transformation to data analytics, and from dashboard to realize business intelligence tools.
- **Living Lab coworking tools**, sharing, and delegating data, resource, IOT devices, IOT Edge, heatmaps, IOT Applications, dashboards, blogs, articles, videos, external services, micro applications, MicroServices, social interaction, etc.;
- **End-2-end encrypted** communication, from devices to dashboard, compliant with GDPR privacy/security, mutual authenticated (or other models), PEN test passed.
- **Ready to use Appliance Virtual Machines and/or Containers** for a modules and tools, free of license, several different configurations from Micro to Large.
- **usable on cloud and on premise with your private installation is 100% open source**, easy to install from VM and Docker Compose which are provided free of license.
- **Modular, secure, elastic scalability and robustness by design** are also provided with automated scaling for IOT applications, Data Analytics, data processing and ingestion.

### V.D. Installation on Premise

Please see: <https://www.snap4city.org/471> for VM

and <https://www.snap4city.org/738> for docker compose



## VI. Selected References

<https://www.snap4city.org/426>

- [What-IF-trafficflow] P. Bellini, S. Bilotta, L. A. Ipsaro Palesi, P. Nesi, G. Pantaleo, "Vehicular Traffic Flow Reconstruction Analysis to Mitigate Scenarios with Large City Changes", IEEE Access, 2022, ISSN: 2169-3536. <https://ieeexplore.ieee.org/document/9984661>
- [TrafficFlowPrediction-DeepLearning] S. Bilotta, E. Collini, P. Nesi, G. Pantaleo, Short-Term Prediction of City Vehicle Flow via Convolutional Deep Learning, IEEE Access, 2022, 10.1109/ACCESS.2022.3217240.
- [DORAM] A. Arman, C. Badii, P. Bellini, S. Bilotta, P. Nesi, M. Paolucci, Analyzing demand with respect to offer of mobility, Applied Science, MDPI, 2022. <https://www.mdpi.com/2076-3417/12/18/8982>
- [EstimatingCO2Emissions] S. Bilotta, P. Nesi, "Estimating CO2 Emissions from IoT Traffic Flow Sensors and Reconstruction", Sensors, MDPI, 2022. <https://www.mdpi.com/1424-8220/22/9/3382/>
- [PredictingLandslideEvents] E. Collini, L. A. Ipsaro Palesi, P. Nesi, G. Pantaleo, N. Nocentini, A. Rosi, "Predicting and Understanding Landslide Events with Explainable AI", IEEE Access, 2022. 10.1109/ACCESS.2022.3158328 <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9732490>
- [IoT-enabledSmartCities] P. Bellini, P. Nesi, G. Pantaleo, "IoT-enabled Smart Cities: a review of concepts, frameworks and key technologies", Applied Science, MDPI, 2022. <https://www.mdpi.com/2076-3417/12/3/1607>
- [MultiClusteringRecommendation] P. Bellini, L. A. Ipsaro Palesi, P. Nesi, G. Pantaleo, "Multi Clustering Recommendation System for Fashion Retail", Multimedia Tools and Applications, Springer, 2022. <https://link.springer.com/article/10.1007/s11042-021-11837-5>
- [AutomatingIoTDataIngestion] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, M. Paolucci, "Automating IoT Data Ingestion Enabling Visual Representation", Sensors, MDPI, 2021. <https://www.mdpi.com/1424-8220/21/24/8429>
- [DigitalTwin1] L. Adreani, P. Bellini, C. Colombo, M. Fanfani, P. Nesi, G. Pantaleo, R. Pisanu, "Digital Twin Framework for Smart City Solutions", DMSVIVA 2022, The 28th International DMS Conference on Visualization and Visual Languages, KSIR Virtual Conference Center, Pittsburgh, USA, June 29 - 30, 2022, <http://ksiresearch.org/seke/dmsviva22.html>, Best Paper Award <https://www.snap4city.org/drupal/sites/default/files/files/DMS-VIVA-3D-DigitalTwin-8pag-final3.pdf>
- [IOT-Directory] P. Bellini, C. Camerota, P. Nesi, "Automating Heterogeneous IoT Device Networks from Multiple Brokers with Multiple Data Models", 2022 Global Internet of Things Summit (GIoTS), Dublin, 20-23 June, 2022, IEEE Press. <https://globaliotsummit.org/> <https://www.snap4city.org/drupal/sites/default/files/files/Automating-IOT-v2-6pagine3-f.pdf>
- [DigitalTwin2] L. Adreani, C. Colombo, M. Fanfani, P. Nesi, G. Pantaleo, R. Pisanu, "Rendering 3D City for Smart City Digital Twin", IEEE SMARTCOMP, June 2022, Aalto University, Espoo, Finland. <https://smartcomp.aalto.fi/>
- [SmartSpecialization] S. Chiordi, G. Desogus, C. Garau, P. Nesi, P. Zamperlin, "A Preliminary Survey on Smart Specialization Platforms: Evaluation of European Best Practices", ICCSA2022, 22nd Int. Conf. on Computational Science and its Applications, Malaga, Spain, July, 2022, LNCS Springer Verlag. <https://sites.unica.it/weaki-transit/2022/01/21/speed2022/> [https://www.snap4city.org/download/video/ICCSA22\\_Chiordietal.pdf](https://www.snap4city.org/download/video/ICCSA22_Chiordietal.pdf)
- [FederatingSmartCities] A. Arman, P. Bellini, P. Nesi, "Searching for Heterogeneous GeoLocated Services via API Federation", ICCSA2022, 22nd Int. Conf. on Computational Science and its Applications, Malaga, Spain, July, 2022, LNCS Springer Verlag. <https://sites.unica.it/weaki-transit/2022/01/21/speed2022/> [https://www.snap4city.org/download/video/LNCS\\_FederatedKB-SuperServiceMap-v0-4.pdf](https://www.snap4city.org/download/video/LNCS_FederatedKB-SuperServiceMap-v0-4.pdf)
- [GeneratingDigitalTwin] L. Adreani, C. Colombo, M. Fanfani, P. Nesi, G. Pantaleo, R. Pisanu, "A Photorealistic 3D City Modeling Framework for Smart City Digital Twin", SCC workshop at IEEE SMARTCOMP, June 2022, Aalto University, Espoo, Finland. <https://smartcomp.aalto.fi/> <https://www.snap4city.org/download/video/SCC2022.pdf>
- [ShortTermPredictionBikeSharing2021] E. Collini, P. Nesi, G. Pantaleo, "Deep Learning for Short-Term Prediction of Available Bikes on Bike-Sharing Stations", under publication on IEEE Access, pp.1-11, Print ISSN: 2169-3536, Online ISSN: 2169-3536, DOI: <https://doi.org/10.1109/ACCESS.2021.3110794>
- [ChemicalPlant2021] P. Bellini, D. Cenni, N. Mitolo, P. Nesi, G. Pantaleo, M. Soderi, "High Level Control of Chemical Plant by Industry 4.0 Solutions", Journal of Industrial Information Integration, Elsevier. <https://doi.org/10.1016/j.jii.2021.100276>
- [LongTermPredictionBikeSharing2021] D. Cenni, E. Collini, P. Nesi, G. Pantaleo, I. Paoli, "Long-Term Prediction of Bikes Availability on Bike-Sharing Stations", Journal of Visual Languages and Computing, KSI. <https://doi.org/10.18293/jvlc2021-n1-001>
- [SatelliteData2021] P. Bellini, D. Cenni, N. Mitolo, P. Nesi, G. Pantaleo, "Exploiting Satellite Data in the Context of Smart City Applications", The 5th IEEE International Conference on Smart City Innovations - Track 1: Theory, Modeling and Methodologies, 2021. <http://ieeesmartworld.org/sci/> <https://www.snap4city.org/drupal/sites/default/files/files/Copernicus-final.pdf>
- [PredictiveMaintenance2021] P. Bellini, D. Cenni, L. A. Ipsaro Palesi, P. Nesi, G. Pantaleo, "A Deep Learning Approach for Short Term Prediction of Industrial Plant Working Status", 7th IEEE International Conference on Big Data Service and Machine Learning Applications, 23-26 August, 2021. <http://www.big-dataservice.net/index.html>
- [15MinCityIndex2021] C. Badii, P. Bellini, D. Cenni, S. Chiordi, N. Mitolo, P. Nesi, M. Paolucci, "Computing 15MinCityIndexes on the basis of Open Data and Services", Proc. of the 2021 International Conference on Computational Science and Its Applications. Published on LNCS Springer. <https://iccsa.org/>

- [https://www.snap4city.org/drupal/sites/default/files/files/computing15minCityIndex\\_ICCSA\\_v0-3.pdf](https://www.snap4city.org/drupal/sites/default/files/files/computing15minCityIndex_ICCSA_v0-3.pdf)  
web page: <https://www.snap4city.org/drupal/node/652>
- [LongTermPredictionNO2-2021] P. Bellini, S. Bilotta, D. Cenni, E. Collini, P. Nesi, G. Pantaleo, M. Paolucci, "Long Term Predictions of NO2 Average Values via Deep Learning", Proc. of the 2021 International Conference on Computational Science and Its Applications. Published on LNCS Springer. <https://iccsa.org/>
- [UserBehaviourModalities2021] C. Badii, A. Difino, P. Nesi, I. Paoli, M. Paolucci, "Classification of Users Transportation Modalities from Mobiles in Real Operating Conditions", Multimedia Tools and Applications, Springer, 2021. <https://doi.org/10.1007/s11042-021-10993-y> or as a PDF here <https://link.springer.com/content/pdf/10.1007/s11042-021-10993-y.pdf>
- [Resilience2021] E. Bellini, P. Bellini, D. Cenni, P. Nesi, G. Pantaleo, I. Paoli, M. Paolucci, "An IoE and Big Multimedia Data approach for Urban Transport System resilience management in Smart City", Sensors, MDPI, 2021, <https://www.mdpi.com/1424-8220/21/2/435/pdf>
- [AnomalyDetection2020] P. Bellini, D. Cenni, P. Nesi, "Anomaly Detection on IOT Data for Smart City", Proc of 6<sup>th</sup> IEEE International Workshop on Sensors and Smart Cities, with IEEE SmartComp, 14-17 Sept. 2020, Bologna, Italy. <http://ssc2020.unime.it/>  
<https://www.snap4city.org/download/video/AnomalyDetection2020.pdf>
- [BigDataService2018] P. Nesi, P. Bellini, M. Paolucci, I. Zaza, "Smart City architecture for data ingestion and analytics: processes and solutions", IEEE BigDataService 2018, Bamberg, Germany, March 26 - 29, 2018.  
<https://www.snap4city.org/download/video/BigDataService2018.pdf>
- [COVID-19] C. Badii, P. Bellini, S. Bilotta, D. Bologna, D. Cenni, A. Difino, A. Ipsaro Palesi, N. Mitolo, P. Nesi, G. Pantaleo, I. Paoli, M. Paolucci, M. Soderi, "How COVID-19 Lockdown Impacted on Mobility and Environmental data", Bollettino della Società Geografica Italiana, FuPress, June 2020, DOI: 10.13128//bsgi.v2i2.932  
<https://drive.google.com/file/d/1hN4dFuEXLMLq4rMY8u88ilZnrVk3HPll/view>  
<https://www.snap4city.org/download/video/COVID-19.pdf>
- [DashboardProduction2020] Q. Han, P. Nesi, G. Pantaleo, I. Paoli, "Smart City Dashboards: Design, Development and Evaluation", Proc. Of the IEEE ICHMS 2020, International Conference on Human Machine Systems, September 2020.  
<http://ichms.dimes.unical.it/>  
<https://www.snap4city.org/download/video/DashboardProduction2020.pdf>
- [Dashboards2019] P. Bellini, D. Cenni, M. Marazzini, N. Mitolo, P. Nesi, M. Paolucci, "Smart City Control Room Dashboards: Big Data Infrastructure, from data to decision support", Journal of Visual Languages and Computing, 10.18293/VLSS2018-030  
<https://ksiresearchorg.ipage.com/vlss/journal/VLSS2018/paper%2030.pdf>  
<https://www.snap4city.org/download/video/Dashboards2019.pdf>
- [DataFlow2019] P. Bellini, F. Bugli, P. Nesi, G. Pantaleo, M. Paolucci, I. Zaza, "Data Flow Management and Visual Analytic for Big Data Smart City/IOT", 19<sup>th</sup> IEEE Int. Conf. on Scalable Computing and Communication, IEEE SCALCOM 2019, Leicester, UK  
<https://www.slideshare.net/paolonesi/data-flow-management-and-visual-analytic-for-big-data-smart-cityiot>  
<https://www.snap4city.org/download/video/DataFlow2019.pdf>
- [DI-DataInspectorBase] P. Bellini, D. Bologna, Q. Han, P. Nesi, G. Pantaleo, M. Paolucci, "Data Ingestion and Inspection for Smart City Applications", proc. Of IEEE International Conference SMARTCOMP 2020, Bologna, Italy, 2020. <http://www.smart-comp.org/>  
<https://www.snap4city.org/download/video/DI-DataInspectorBase.pdf>
- [DORAM2020] A. Arman, P. Bellini, P. Nesi, M. Paolucci, "Analysing Public Transportation Offer wrt Mobility Demand", ACM Workshop on Technology Enablers and Innovative Applications for Smart Cities and Communities (TESCA 2019), November 10, 2019 at Columbia University, New York, USA. <https://dl.acm.org/citation.cfm?id=3364828>  
<https://doi.org/10.1145/3364544.3364828>  
[https://dl.acm.org/ft\\_gateway.cfm?id=3364828&ftid=2101673&dwn=1&CFID=180...](https://dl.acm.org/ft_gateway.cfm?id=3364828&ftid=2101673&dwn=1&CFID=180...)  
<https://www.snap4city.org/download/video/DORAM2020.pdf>
- [FederatedKnowledgeBase2020] P. Bellini, D. Nesi, P. Nesi, M. Soderi, "Federation of Smart City Services via APIs", Proc of 6<sup>th</sup> IEEE International Workshop on Sensors and Smart Cities, with IEEE SmartComp, 14-17 Sept. 2020, Bologna, Italy.  
<http://ssc2020.unime.it/>  
<https://www.snap4city.org/download/video/FederatedKnowledgeBase2020.pdf>
- [Gov2018] M. Azzari, C. Garau, P. Nesi, M. Paolucci, P. Zamperlin, "Smart City Governance Strategies to better move towards a Smart Urbanism", The 18<sup>th</sup> International Conference on Computational Science and Its Applications (ICCSA 2018), July 2 – 5, 2018 in Melbourne, with the Monash University, Australia.  
<https://www.snap4city.org/download/video/Gov2018.pdf>
- [Gov2020] C. Garau, P. Nesi, I. Paoli, M. Paolucci, P. Zamperlin, A Big Data Platform for Smart and Sustainable Cities: Environmental Monitoring case studies in Europe. Proc. Of International Conference on Computational Science and its Applications, ICCSA2020. Cagliari, Italy, 1-4 July 2020. <http://www.iccsa.org/> [https://link.springer.com/chapter/10.1007%2F978-3-030-58820-5\\_30](https://link.springer.com/chapter/10.1007%2F978-3-030-58820-5_30)  
<https://www.snap4city.org/download/video/Gov2020.pdf>
- [Heatmap2020] C. Badii, S. Bilotta, D. Cenni, A. Difino, P. Nesi, I. Paoli, M. Paolucci, "High Density Real-Time Air Quality Derived Services from IoT Networks", Sensors, MDPI, 2020. <https://www.mdpi.com/1424-8220/20/18/5435/htm>  
<https://www.snap4city.org/download/video/Heatmap2020.pdf>

- [Industry4.0-2020] C. Badii, P. Bellini, D. Cenni, N. Mitolo, P. Nesi, G. Pantaleo, M. Soderi, "Industry 4.0 Synoptics Controlled by IoT Applications in Node-RED", Proc. Of the IEEE iThings, Nov 02-06, 2020.  
<https://www.snap4city.org/download/video/Industry40-2020.pdf>
- [KnowledgeBase2018] P. Bellini, P. Nesi, "Performance Assessment of RDF Graph Databases for Smart City Services", Journal of Visual Language and Computing, Elsevier, 2018. <https://doi.org/10.1016/j.jvlc.2018.03.002>  
<https://www.snap4city.org/download/video/KnowledgeBase2018.pdf>
- [LivingLab2018] P. Nesi, M. Paolucci, "Supporting Living Lab with Life Cycle and Tools for Smart City Environments", The 24<sup>th</sup> International DMS Conference on Visualization and Visual Languages, DMSVIVA 2018, Hotel Pullman, Redwood City, San Francisco Bay, California, USA, June 29 – 30, 2018  
<https://www.snap4city.org/download/video/LivingLab2018.pdf>
- [MicroServices2019] C. Badii, P. Bellini, A. Difino, P. Nesi, G. Pantaleo, M. Paolucci, "MicroServices Suite for Smart City Applications", Sensors, MDPI, 2019. <https://doi.org/10.3390/s19214798>  
<https://www.snap4city.org/download/video/MicroServices2019.pdf>
- [MobileDevKit2017] C. Badii, P. Bellini, P. Nesi, M. Paolucci, "A Smart City Development kit for designing Web and Mobile Apps", IEEE international Conference on Smart City and Innovation, 2017, San Francisco. <https://www.slideshare.net/paolonesi/a-smart-city-development-kit-for-designing-web-and-mobile-apps>  
<https://www.snap4city.org/download/video/MobileDevKit2017.pdf>
- [MobilityMicroServices2019] C. Badii, P. Bellini, A. Difino, P. Nesi, "Sii-Mobility: an IOT/IOE architecture to enhance smart city services of mobility and transportation", Sensors, MDPI, 2019. <https://doi.org/10.3390/s19010001>  
<https://www.mdpi.com/1424-8220/19/1/1/pdf>  
<https://www.snap4city.org/download/video/MobilityMicroServices2019.pdf>
- [NOX prediction 2020] C. Badii, S. Bilotta, D. Cenni, A. Difino, P. Nesi, G. Pantaleo, I. Paoli, M. Paolucci, "Real-Time Automatic Air Pollution Services from IOT Data Network", proc. Of IEEE Symposium on Computers and Communications (ISCC), MOCS track, 10<sup>th</sup> Workshop on Management of Cloud and Smart City System, 2020 July 7<sup>th</sup>, Rennes, France.  
<https://conferences.imt-atlantique.fr/iscc2020/>  
<https://www.snap4city.org/download/video/NOXprediction 2020.pdf>
- [Resilience2017] E. Bellini, P. Ceravolo, P. Nesi, "Quantify resilience enhancement of UTS through exploiting connected community and internet of everything emerging technologies", 2017, <http://hdl.handle.net/2158/1105460>, ACM TRANSACTIONS ON INTERNET TECHNOLOGY <https://dl.acm.org/citation.cfm?id=3137572>  
<https://www.snap4city.org/download/video/Resilience2017.pdf>
- [Resilience2019] E. Bellini, L. Cocone, P. Nesi, "A Functional Resonance Analysis Method driven Resilience Quantification for socio-technical System", IEEE Systems Journal, DOI 10.1109/JSYST.2019.2905713, ISSN: 1932-8184, ISSN: 1937-9234, pp.1-11, 2019 <https://ieeexplore.ieee.org/document/8686178>  
<https://www.snap4city.org/download/video/Resilience2019.pdf>
- [Scalable2018] C. Badii, E. G. Belay, P. Bellini, D. Cenni, M. Marazzini, M. Mesiti, P. Nesi, G. Pantaleo, M. Paolucci, S. Valtolina, M. Soderi, I. Zaza, "Snap4City: A Scalable IOT/IOE Platform for Developing Smart City Applications", Int. Conf. IEEE Smart City Innovation, Cina 2018, IEEE Press. <https://ieeexplore.ieee.org/document/8560331/>  
<https://www.snap4city.org/download/video/Scalable2018.pdf>
- [SCAPI-KB2017] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, "Analysis and Assessment of a Knowledge Based Smart City Architecture Providing Service APIs", Future Generation Computer Systems, Elsevier, 2017, <http://dx.doi.org/10.1016/j.future.2017.05.001>  
<https://www.snap4city.org/download/video/SCAPI-KB2017.pdf>
- [Security2020] C. Badii, P. Bellini, A. Difino, P. Nesi, "Smart City IoT Platform Respecting GDPR Privacy and Security Aspects", IEEE Access, 2020. 10.1109/ACCESS.2020.2968741 <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8966344>  
<https://www.snap4city.org/download/video/Security2020.pdf>
- [SmartDS] M. Bartolozzi, P. Bellini, P. Nesi, G. Pantaleo and L. Santi, "A Smart Decision Support System for Smart City," 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, 2015, pp. 117-122, December 2015, Cina, IEEE press, doi: 10.1109/SmartCity.2015.57  
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7463711&isnumber=7463653>  
<https://www.snap4city.org/download/video/SmartDS.pdf>
- [SmartParking2018] C. Badii, P. Nesi, I. Paoli, "Predicting available parking slots on critical and regular services exploiting a range of open data", IEEE Access, 2018, <https://ieeexplore.ieee.org/abstract/document/8430514/>  
<https://www.snap4city.org/download/video/SmartParking2018.pdf>
- [TrafficFlowReconstruction2020] S. Bilotta, P. Nesi, "Traffic Flow Reconstruction by Solving Indeterminacy on Traffic Distribution at Junctions", Future Generation Computer Systems, Elsevier, 2020, 2021. [https://authors.elsevier.com/sd/article/S0167-739X\(20\)30835-9](https://authors.elsevier.com/sd/article/S0167-739X(20)30835-9)  
<https://www.snap4city.org/download/video/TrafficFlowReconstruction2020.pdf>
- [TrafficPredicitons2020] S. Bilotta, P. Nesi, I. Paoli, "Real-Time System for Short- and Log-Term Prediction of Vehicle Flow", IEEE Smart Data Service as Proc of IEEE Services, 2020.  
<https://www.snap4city.org/download/video/TrafficPredicitons2020.pdf>

- [TwitterVigilance2017] D. Cenni, P. Nesi, G. Pantaleo, I. Zaza, "Twitter Vigilance: a Multi-User platform for Cross-Domain Twitter Data Analytics, NLP and Sentiment Analysis", IEEE international Conference on Smart City and Innovation, 2017, San Francisco. <https://www.slideshare.net/paolonesi/twitter-vigilance-a-multiuser-platform-for-crossdomain-twitter-data-analytics-nlp-and-sentiment-analysis>  
<https://www.snap4city.org/download/video/TwitterVigilance2017.pdf>
- [UserBehavior2020] C. Badii, A. Difino, P. Nesi, I. Paoli, M. Paolucci, "Classification of Users' Transportation Modalities in Real Conditions", Proc. Of the 26<sup>th</sup> International DMS Conference on Visualization and Visual Languages, DMSVIVA, Wyndham Pittsburgh University Center, Pittsburg, USA, July 7-8, 2020. <http://ksiresearch.org/seke/dmsviva20.html>  
<https://www.snap4city.org/download/video/UserBehavior2020.pdf>
- [UserBehaviour2017] P. Bellini, D. Cenni, P. Nesi, I. Paoli, "Wi-Fi Based City Users' Behaviour Analysis for Smart City", Journal of Visual Language and Computing, Elsevier, 2017. <http://www.sciencedirect.com/science/article/pii/S1045926X17300083>  
<https://www.snap4city.org/download/video/UserBehaviour2017.pdf>
- [UserEngagement2017] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, "User Engagement Engine for Smart City Strategies", IEEE International Conference on Smart Computing, IEEE SMARTCOMP 2017, Hong Kong.  
<https://www.snap4city.org/download/video/UserEngagement2017.pdf>
- [Resilience2023] Bellini, E., Nesi, P., Martelli, C., Gaitanidou, E., Archetti, F., Candelieri, A., ... & Cocone, L. (2023). Building Resilient and Sustainable Cities Starting from the Urban Transport System. In *Urban Resilience: Methodologies, Tools and Evaluation: Theory and Practice* (pp. 49-74). Cham: Springer International Publishing.
- [Resilience2017b] E. Bellini, P. Nesi, Exploiting smart technologies to build Smart Resilient Cities, *Routledge Handbook of Sustainable and Resilient Infrastructure*, 685-705
- [Resilience2016] Emanuele Bellini, Paolo Nesi and Pedro Ferreira, "Operationalize Data-driven Resilience in Urban Transport Systems", part of the IRGC Resource Guide on Resilience, available at: <https://www.irgc.org/risk-governance/resilience/>. IRGC (2016). Lausanne: EPFL International Risk Governance Center. v29-07-2016

## VI.A. Training Course on Slides and Interactive Slides, Videos

**TECHNICAL OVERVIEW:** <https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>

**This document:** <https://www.snap4city.org/download/video/Snap4Tech-Development-Life-Cycle.pdf>

|                                              | 1st part | 2nd part   | 3rd part             | 4th part       | 5th part                 | 6th part                  | 7th part                       | 8th                                |
|----------------------------------------------|----------|------------|----------------------|----------------|--------------------------|---------------------------|--------------------------------|------------------------------------|
| what                                         | Overview | Dashboards | IoT App, IOT Network | Data Analytics | Data Ingestion processes | System and Deploy Install | Smart City API: Web & Mob. App | Design and Develop Smart Solutions |
| PDF 2022                                     |          |            |                      |                |                          |                           |                                |                                    |
| Interactive (2022) with video and animations |          |            |                      |                |                          |                           |                                |                                    |

|                                                      |                                                                                                                                                                                                                                   |
|------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Training Course page                                 | <a href="https://www.snap4city.org/577">https://www.snap4city.org/577</a><br>use this link to get updated versions of the following training sections and interactive versions cited in the paper.                                |
| Training Part 1 - overview                           | <a href="https://www.snap4city.org/download/video/course/gen/Snap4City-1st-Overview-v6-0.pdf">https://www.snap4city.org/download/video/course/gen/Snap4City-1st-Overview-v6-0.pdf</a>                                             |
| Training Part 2 - Dashboard                          | <a href="https://www.snap4city.org/download/video/course/das/Snap4City-2nd-slot-dashboard-building-v10-4.pdf">https://www.snap4city.org/download/video/course/das/Snap4City-2nd-slot-dashboard-building-v10-4.pdf</a>             |
| Training Part 3 – IoT App and networks               | <a href="https://www.snap4city.org/download/video/course/iot/Snap4City-3rd-slot-IOT-Applications-v10-0.pdf">https://www.snap4city.org/download/video/course/iot/Snap4City-3rd-slot-IOT-Applications-v10-0.pdf</a>                 |
| Training Part 4 – Data Analytics                     | <a href="https://www.snap4city.org/download/video/course/da/Snap4City-4th-slot-Data-Analytic-v9-0.pdf">https://www.snap4city.org/download/video/course/da/Snap4City-4th-slot-Data-Analytic-v9-0.pdf</a>                           |
| Training Part 5 – Data Ingestion                     | <a href="https://www.snap4city.org/download/video/course/di/Snap4City-5th-slot-data-ingestion-v8-9.pdf">https://www.snap4city.org/download/video/course/di/Snap4City-5th-slot-data-ingestion-v8-9.pdf</a>                         |
| Training Part 6 – System deploy, summary             | <a href="https://www.snap4city.org/download/video/course/sys/Snap4City-6th-slot-system-deploy-v8-7.pdf">https://www.snap4city.org/download/video/course/sys/Snap4City-6th-slot-system-deploy-v8-7.pdf</a>                         |
| Training Part 7 – API and Mobile Apps                | <a href="https://www.snap4city.org/download/video/course/app/Snap4City-7th-slot-SCAPI-Mobile-Applications-v5-0.pdf">https://www.snap4city.org/download/video/course/app/Snap4City-7th-slot-SCAPI-Mobile-Applications-v5-0.pdf</a> |
| Training Part 8 – Design and Develop Smart Solutions | <a href="https://www.snap4city.org/download/video/course/dev/Snap4City-8th-Solutions-Development-v9-6.pdf">https://www.snap4city.org/download/video/course/dev/Snap4City-8th-Solutions-Development-v9-6.pdf</a>                   |
| Snap4City Impact Story at FIWARE                     | <a href="https://www.snap4city.org/drupal/sites/default/files/files/FF_ImpactStories_Snap4City.pdf">https://www.snap4city.org/drupal/sites/default/files/files/FF_ImpactStories_Snap4City.pdf</a>                                 |



## VII. Appendix: Data Dictionary for Entity Models (IoT Device Models) and Entity Instances (Devices), Entity Variables, attributes and metrics

Please note that this is just an example, since new definitions are frequently updated. For any doubts please refer to the Entity/IoT Directory tool in which you can actually define the values.

Please note that: in the definition of Entity Model (IoT Device Model), you have to define for each variable (Entity Variable, values) the Value Name, Value\_Type, Value\_Unit and Data\_Type, and for this purpose you have to use the Dictionary reported in section VII Appendix data dictionary. The Entity Directory (IoT Directory) has an on-line dictionary directly enforced into the assisted tool for defining the Entity Models (IoT Device Models). An online version of the table in appendix can be recovered from <https://www.snap4city.org/818>

Any update and addition to the dictionary of snap4city.org has to be requested to snap4city@disit.org. If you have your own instance of the platform, you can define your own dictionary and request a copy of the snap4city.org dictionary by email. The dictionary is used into the IoT Device Model definition, in mapping smart data models, and in creating full custom devices.

Please note that Entity Models/Instances have to follow the rules:

- **ValueName** is the name of the variable/values,
  - it does not accept any space or special char in its definition as in most of the programming languages.
  - for each Value Name you have to specify **Value\_Type**, **Value\_Unit** and **Data\_Type**,
  - **Value Name cannot be one of the strings used for Value\_Type, Value\_Unit and Data\_Type**
  - **see** Section VII Appendix Data Dictionary.
- according to the **Value\_Type** selected the possible values for
  - **Value\_Unit** would be restricted. These constraints are directly applied in the user interface tool of the Entity Directory (IoT Directory).
  - **Data\_Type** would be restricted. These constraints are directly applied in the user interface tool of the Entity Directory (IoT Directory).
- **Value\_Type, Value\_Unit, Data\_Type** cannot have spaces in their name definitions.
- **Data\_Type** for **Value\_Unit** defined as timestamps are strings. They are expected to be formatted as ISOString: YYYY-MM-DDTHH:mm:ss.sssZ which can be obtained by using function toISOString() of JavaScript on variables of kind Date()
  - Please note that ISOstrings have to be in UTC to avoid misplacement of events coming in different time regions in the wrong position, and thus to reproduce wrong visualization once you change the international time fuse.
  - Some examples:
    - "2020-08-04T04:00:00+02:00",
    - "2020-08-03T00:00:00.000Z"
  - Some code to get it:
    - ```
var strnow = new Date().toISOString();
```
 - for fuse management
 - ```
var todaynow = new Date();
dateCET2Z(todaynow).toISOString();

function dateCET2Z(date) {
```

```
d = new Date(date).toLocaleString('nl-BE', {timeZone: 'Europe/Brussels'});
offset = new Date(d).getTime() - new Date(date).getTime();
return new Date(new Date(date).getTime() - offset);
}
```

- Time **Durations** can be in milliseconds, hours, minutes, etc... and are typically integer or float.
- Each Entity Model and Entity Instance may have
  - **At most one** ValueName defined as **dateObserved**, with value\_type = timestamp.
  - **At most one** Value Type defined as Timestamp
  - **dateObserved** should have **Value\_Type** defined as **Timestamp** to create Time Series
  - any other variable, and thus Value Name which needs to have as Value\_Unit of timestamp in millisecond has to provide a **Value\_Type** defined as **Datetime** which is also coded in time stamp millisecond and in ISO string.
  - The **absence of unique single Value\_Type per model/entity** defined as **Timestamp** would bring to create a stable Entity without time series.
  - The **presence of multiple Value\_Type per model/entity** defined as **Timestamp** would bring to create unpredictable behavior.
  - **forbidden** ValueNames are: type, id, value;
- **Data\_Types** are typically:
  - **integer, float (AKA numeric).** Any variable on which you would like to apply math operators on queries (and thus on the corresponding Smart City API) such as ==, <, >, <=, >=, etc. has to be defined with a Data\_Type which is numeric. Integer and Float do not have limitations on their dynamics.
  - **String.** Any variable can be also defined and loaded with strings, and you can send on strings also numbers. The only operator on strings is the verification of the equality ==. Good for status detection.
  - **JSON.** Can be JSON data structure.
- **JSON Data\_type** have some limitations. Since if your have JSON data type:
  - The loaded JSON pack will be stored as a string into the Storage
  - The loaded JSON data will not be
    - singularly indexed into the storage, so that they will not be automatically usable into Time Series, and dashboards for showing values, sequences, barseries, spidernet, etc.
    - search-able by using queries and query by value of the Smart City API

| Value Type                       | Description                      | possible Value Units | Possible Data Types |
|----------------------------------|----------------------------------|----------------------|---------------------|
| actuator_canceller               | Actuator Canceller               |                      | string              |
| actuator_deleted                 | Actuator Deleted                 |                      | integer             |
| actuator_deletion_date           | Actuator Deletion Date           | timestamp            | string              |
| air_quality_index                | Air quality index                |                      | float               |
| altitude                         | Altitude                         | m                    | float, integer      |
| angle                            | angle                            | deg                  | float               |
| annual_C6H6_average              | annual_C6H6_average              | ppm, mg/m3, µg/m³    | float               |
| annual_C6H6_exceedance_count     | annual_C6H6_exceedance_count     | #                    | integer, float      |
| annual_CO_average                | annual_CO_average                | ppm, mg/m3, µg/m³    | float               |
| annual_CO_exceedance_count       | annual_CO_exceedance_count       | #                    | integer, float      |
| annual_NO2_average               | annual_NO2_average               | ppm, mg/m3, µg/m³    | float               |
| annual_NO2_exceedance_count      | annual_NO2_exceedance_count      | #                    | integer, float      |
| annual_O3_average                | annual_O3_average                | ppm, mg/m3, µg/m³    | float               |
| annual_O3_exceedance_count       | annual_O3_exceedance_count       | #                    | integer, float      |
| annual_particle_average          | annual_particle_average          | ppm, mg/m3, µg/m³    | float               |
| annual_particle_exceedance_count | annual_particle_exceedance_count | #                    | integer, float      |
| annual_PM10_average              | annual_PM10_average              | ppm, mg/m3, µg/m³    | float               |

|                                   |                                      |                                                 |                      |
|-----------------------------------|--------------------------------------|-------------------------------------------------|----------------------|
| annual_PM10_exceedance_count      | Annual PM10 Exceedance Count         | #                                               | integer,float        |
| annual_PM2_5_average              | annual_PM2_5_average                 | ppm, mg/m <sup>3</sup> , µg/m <sup>3</sup>      | float                |
| annual_PM2_5_exceedance_count     | annual_PM2_5_exceedance_count        | #                                               | integer,float        |
| anomaly_level                     | anomaly level for traffic            | -                                               | string,integer,float |
| asleep_time                       | Asleep Time                          | min                                             | string               |
| audio                             | Audio                                |                                                 | string,float         |
| available_bikes                   | Available Bikes                      | #                                               | integer              |
| average_accelerometric_intensity  | Average Accelerometric Intensity     | mV, m/s <sup>2</sup>                            | float                |
| average_atmospheric_pressure      | Average Atmospheric Pressure         | hPa, bars                                       | float                |
| average_brightness                | Average Brightness                   | lux                                             | float                |
| average_heart_rate                | Average Heart Rate                   | bpm                                             | float                |
| average_humidity                  | Average Humidity                     | kg/m <sup>3</sup>                               | float                |
| average_noise                     | Average Noise                        | dB                                              | float                |
| average_respiratory_matrix_signal | Average Respiratory Matrix Signal    | bpm                                             | float                |
| average_respiratory_rate          | Average Respiratory Rate             | bpm                                             | float                |
| average_temperature               | Average Temperature                  | °C                                              | float                |
| average_vehicle_distance          | Average Vehicle Distance             | Km, m                                           | float                |
| average_vehicle_speed             | Average Vehicle Speed                | km/h, m/s                                       | float                |
| average_vehicle_time              | Average Vehicle Time                 | s                                               | string,float         |
| battery_level                     | Battery Level                        | %                                               | float                |
| BC_concentration                  | Bc Concentration                     | ppb, ppm                                        | float                |
| benzene_concentration             | Benzene Concentration                | ppb, ppm, µg/m <sup>3</sup>                     | float                |
| bike_count                        | number of bikes                      | K#, #                                           | integer              |
| blue_code_count                   | Blue Code Count                      | #                                               | integer              |
| brightness_flag                   | Brightness Flag                      | #                                               | string               |
| broken_bikes                      | Broken Bikes                         | #                                               | integer              |
| burning_hours                     | burning_hours                        | hours                                           | string               |
| button                            | Button                               | #                                               | integer              |
| Capacity                          | Volume capacity                      | l                                               | float                |
| car_park_exit_rate                | Car Park Exit Rate                   |                                                 | float                |
| car_park_fill_rate                | Car Park Fill Rate                   |                                                 | float                |
| car_park_free_places              | Car Park Free Places                 | #                                               | integer              |
| car_park_occupancy                | Car Park Occupancy                   | %                                               | string,float         |
| car_park_occupancy_time           | Car Park Occupancy Time              | min                                             | string               |
| car_park_occupied_places          | Car Park Occupied Places             | #                                               | integer              |
| car_park_status                   | Car Park Status                      | status                                          | string               |
| car_park_validity_status          | Car Park Validity Status             | status                                          | string               |
| car_plate                         | Car plate                            | targa                                           | string               |
| charging_level                    | Charging Level                       | %                                               | float                |
| charging_state                    | Charging State                       | status                                          | string               |
| charging_station_state            | Charging Station State               | status                                          | string               |
| cloud_cover                       | Cloud cover                          | -, %                                            | float                |
| CO2_concentration                 | CO2 Concentration                    | ppb, ppm, µg/m <sup>3</sup>                     | float                |
| CO_concentration                  | CO Concentration                     | ppb, ppm, mg/m <sup>3</sup> , µg/m <sup>3</sup> | float                |
| Count                             | count                                | H#, M#, K#, #                                   | integer              |
| creation_date                     | Creation Date                        | timestamp                                       | string               |
| current                           | Current                              | A, mA, KA                                       | float                |
| current_working_mode              | current_working_mode                 |                                                 | string,integer       |
| CylinderSize                      | Size of a Cylinder for gas or liquid | l                                               | float                |
| daily_O3_exceedance_count         | Daily O3 Exceedance Count            | µg/m <sup>3</sup> , #                           | integer              |
| dali_com_error                    | dali_com_error                       | bool                                            | string               |
| dali_dimming_error                | dali_dimming_error                   | bool                                            | string               |
| dali_gear_error                   | dali_gear_error                      | bool                                            | string               |
| dali_lamp_error                   | dali_lamp_error                      | bool                                            | string               |
| datastructure                     | data structure                       | complex                                         | json                 |
| DataTransferred                   | Transferred Data                     | Kbyte                                           | float                |
| date                              | Date                                 | timestamp                                       | string               |
| datetime                          | Datetime                             | timestamp                                       | string               |
| description                       | Entity Description                   | text                                            | string               |
| dew_point                         | Dew Point                            | °C                                              | float                |
| dimension                         | measurable dimension                 | mm, cm, mt                                      | integer,float        |
| displacement                      | Displacement of a Motor              | cm <sup>3</sup> , l                             | float                |
| distance                          | Distance                             | Km, m                                           | integer,float        |

|                                   |                                   |                                    |                |
|-----------------------------------|-----------------------------------|------------------------------------|----------------|
| duration                          | Duration                          | s, min, hours, day, month, year    | integer,float  |
| electro_conductivity              | Electro Conductivity              | mS/cm, $\mu$ S/cm                  | float          |
| electro_valve_action              | Electro Valve Action              |                                    | string         |
| energy                            | Energy                            | KW/h, MW/H, wh                     | float          |
| entity_creator                    | Entity Creator                    |                                    | string         |
| entity_desc                       | Entity Desc                       |                                    | string         |
| enviromental_quality_flag         | Enviromental Quality Flag         | #                                  | string         |
| fan                               | Fan                               | -                                  | integer,float  |
| fast_charging_status              | Fast Charging Status              | status                             | string         |
| Flow_of_Gas_as_Metane             | Flow of Gas as Metane             | SMC                                | float          |
| free_stalls                       | Free Stalls                       | #                                  | integer        |
| freeze                            | Freeze                            |                                    | string,float   |
| fuel_price                        | Fuel Price                        | euro                               | float          |
| fuel_type                         | Fuel Type                         |                                    | string         |
| GEI                               | Guest Experience Index            | #                                  | float          |
| geolocation                       | geolocation                       | text                               | string         |
| glucose_percentage                | Glucose Percentage                | %                                  | float          |
| green_code_count                  | Green Code Count                  | #                                  | integer        |
| H2S_concentration                 | H2S Concentration                 | ppb, ppm, $\mu$ g/m <sup>3</sup>   | float          |
| hailDensity                       | hailAmount per area               | hits/cm2                           | float          |
| height                            | height or quote                   | m                                  | float          |
| high                              | height                            |                                    | float          |
| hour_O3_max                       | Hour O3 Max                       | $\mu$ g/m <sup>3</sup>             | float          |
| humidity                          | Humidity                          | %                                  | float          |
| humidity_flag                     | Humidity Flag                     | #                                  | string         |
| Identifier                        | Identifier                        | ID, SURI                           | integer,string |
| image                             | image                             | imagebuffer                        | string         |
| ir                                | Ir                                |                                    | string,float   |
| lamp_level                        | Lamp Level                        | %                                  | float          |
| lamp_temperature                  | Lamp Temperature                  | °C                                 | float          |
| latitude                          | Latitude                          | deg                                | float          |
| latitude_longitude                | Latitude Longitude                | latlon                             | string,json    |
| leaf_wetness                      | Leaf Wetness                      | %                                  | string         |
| light                             | Light                             | lux                                | float          |
| light_intensity                   | Light Intensity                   | lux, %                             | float          |
| light_level                       | Light Intensity                   | %                                  | float          |
| lightpoint_status                 | lightpoint_status                 |                                    | string         |
| likertvote                        | vote in likert scale              | vote                               | integer        |
| longitude                         | Longitude                         | deg                                | float          |
| Matter_over_time                  | Average Respiratory Matrix Signal | t/h, t/d                           | string         |
| max_temperature                   | Max Temperature                   | °C                                 | float          |
| message                           | Message                           |                                    | string         |
| min_temperature                   | Min Temperature                   | °C                                 | float          |
| monitor_status                    | Monitor Status                    | status                             | string         |
| monthly_C6H6_average              | monthly_C6H6_average              | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_C6H6_exceedance_count     | monthly_C6H6_exceedance_count     | #                                  | integer        |
| monthly_CO_average                | monthly_CO_average                | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_CO_exceedance_count       | monthly_CO_exceedance_count       | #                                  | integer,float  |
| monthly_NO2_average               | monthly_NO2_average               | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_NO2_exceedance_count      | monthly_NO2_exceedance_count      | #                                  | integer        |
| monthly_O3_average                | monthly_O3_average                | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_O3_exceedance_count       | monthly_O3_exceedance_count       | #                                  | integer        |
| monthly_particle_average          | monthly_particle_average          | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_particle_exceedance_count | monthly_particle_exceedance_count | #                                  | integer,float  |
| monthly_PM10_average              | monthly_PM10_average              | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_PM10_exceedance_count     | monthly_PM10_exceedance_count     | #                                  | integer        |
| monthly_PM2_5_average             | monthly_PM2_5_average             | ppm, mg/m3, $\mu$ g/m <sup>3</sup> | float          |
| monthly_PM2_5_exceedance_count    | monthly_PM2_5_exceedance_count    | #                                  | integer        |
| moonillumination                  | illumination of the moon          | %                                  | float          |
| moonphase                         | Phase of the Moon                 | status                             | string         |

|                                  |                                  |                               |                       |
|----------------------------------|----------------------------------|-------------------------------|-----------------------|
| moonrise_time                    | Moonrise Time                    | HH:MM                         | string                |
| moonset_time                     | Moonset Time                     | HH:MM                         | string                |
| motion_detection                 | Motion Detection                 |                               | string                |
| name                             | just the name                    | text                          | string                |
| NO2_concentration                | NO2 Concentration                | ppb, ppm, ug/m3, µg/m³        | float                 |
| NO_concentration                 | NO Concentration                 | ppb, ppm, ug/m3, µg/m³        | float                 |
| noise_flag                       | Noise Flag                       | #                             | string                |
| noise_laeq                       | Noise LA eq                      | dBA                           | float                 |
| noise_lamax                      | Noise LA max                     | dBA                           | float                 |
| nonrem_time_wrt_total_bed_time   | Nonrem Time Wrt Total Bed Time   | %                             | string                |
| nonrem_time_wrt_total_sleep_time | Nonrem Time Wrt Total Sleep Time | %                             | string                |
| O3_concentration                 | O3 Concentration                 | ppb, ppm, ug/m3, µg/m³        | float                 |
| orientation                      | Orientation                      |                               | string,float          |
| people_count                     | People Count                     | Mean#, H#, M#, K#, #          | integer               |
| people_percentage                | people percentage                | %                             | float                 |
| perc_of_presences                | percentage of presences          | %                             | float                 |
| perceived_temperature            | Perceived Temperature            | °C                            | float                 |
| percentage_variation             | variation                        | %                             | float                 |
| period                           | period                           | text                          | integer               |
| pH                               | value of pH                      | #                             | float                 |
| PM10_concentration               | PM10 Concentration               | ppb, ppm, ug/m3, mg/m3, µg/m³ | float                 |
| PM1_Concentration                | PM1 Concentration                | ppm, ug/m3, mg/m3, µg/m³      | float                 |
| PM2.5_concentration              | PM2.5 Concentration              | ppb, ppm, ug/m3, mg/m3, µg/m³ | float                 |
| pollen_concentration_level       | Pollen Concentration Level       | ppm, mg/m3                    | float                 |
| pollen_concentration_trend       | Pollen Concentration Trend       | ppm, mg/m3                    | float                 |
| pollen_concentration_value       | Pollen Concentration Value       | ppm, mg/m3                    | float                 |
| position                         | a position                       | coord                         | float                 |
| power                            | Power                            | W, MW, KW, GW, milliwatt      | float                 |
| power_meter_m                    | Power Meter M                    |                               | float                 |
| power_meter_s                    | Power Meter S                    |                               | float                 |
| power_state                      | power_state                      |                               | string,float          |
| powerfactor                      | powerfactor                      | #                             | float                 |
| precipitation_type               | Precipitation Type               |                               | string                |
| presence_detection_e             | Presence Detection E             |                               | string                |
| pressure                         | Pressure                         | hPa, bars                     | float                 |
| price                            | price                            | euro, Meuro, keuro            | float                 |
| Radiatio_Flux                    | Watt on square meter             | W/m2                          | float                 |
| rain                             | Rain                             | mm                            | float                 |
| red_code_count                   | Red Code Count                   | #                             | integer               |
| rem_sleep_latency                | Rem Sleep Latency                | min                           | float                 |
| rem_time_wrt_total_bed_time      | Rem Time Wrt Total Bed Time      | %                             | string                |
| rem_time_wrt_total_sleep_time    | Rem Time Wrt Total Sleep Time    | %                             | string                |
| road_condition                   | Road Condition                   |                               | string                |
| salt_concentration               | Salt Concentration               | %                             | float                 |
| SAScore                          | Sentiment Analysis Score         | #                             | float                 |
| sittings_count                   | Sittings Count                   | #                             | integer               |
| sleep_efficiency                 | Sleep Efficiency                 | %                             | float                 |
| sleep_quality_index              | Sleep Quality Index              | #                             | float                 |
| snow                             | Snow                             | cm, mt                        | float                 |
| SO2_concentration                | SO2 Concentration                | ppb, ppm, µg/m³               | float                 |
| SO_concentration                 | SO Concentration                 | ppb, ppm, µg/m³               | float                 |
| soil_humidity                    | Soil Humidity                    | %                             | float                 |
| soil_temperature                 | Soil Temperature                 | °C                            | float                 |
| soil_water_potential             | Soil Water Potential             | cbar                          | float                 |
| solar_radiation                  | photosynthetic radiation         | umol/m2                       | float                 |
| sound_lv                         | Sound Lv                         |                               | string,float          |
| speed                            | Speed                            | m/s                           | float                 |
| state_count                      | State Count                      | #                             | integer               |
| state_time                       | State Time                       |                               | string                |
| status                           | Status                           | status                        | string,integer ,float |



|                                    |                                                                        |                   |               |
|------------------------------------|------------------------------------------------------------------------|-------------------|---------------|
| stop                               | Stop                                                                   |                   | integer       |
| sun_max_height                     | Sun Max Height                                                         |                   | float         |
| sun_max_height_hour                | Sun Max Height Hour                                                    |                   | string        |
| sunrise_time                       | Sunrise Time                                                           | HH:MM             | string        |
| sunset_time                        | Sunset Time                                                            | HH:MM             | string        |
| temperature                        | Temperature                                                            | °K, °F, °C        | float         |
| temperature_flag                   | Temperature Flag                                                       | °K, °F, #         | string        |
| time                               | Time                                                                   | s                 | string        |
| timestamp                          | Timestamp                                                              | timestamp         | string        |
| TOC                                | TOC Density, Title of Concentration                                    | µg/Liter          | float         |
| total_sleep_time                   | Total Sleep Time                                                       | min               | string        |
| traffic_congestion                 | the ration from road capacity in car/h and the effective flow in car/h | -                 | float         |
| TransferRate                       | TransferRate                                                           | Kbps, KByteps     | float         |
| transits_count                     | Transits Count                                                         | #                 | integer       |
| TRSC_concentration                 | TRSC, Total Reduced Sulfur Compounds, Concentration                    | ugS/m3, ug/m3     | float         |
| URI                                | URI LINK                                                               | SURI              | string        |
| URL                                | URL link                                                               | SURI              | string        |
| uv                                 | Uv                                                                     | UVindexUnit       | string,float  |
| variance_accelerometric_intensity  | Variance Accelerometric Intensity                                      | mV                | float         |
| variance_atmospheric_pressure      | Variance Atmospheric Pressure                                          | hPa, bars         | float         |
| variance_brightness                | Variance Brightness                                                    | lux               | float         |
| variance_humidity                  | Variance Humidity                                                      | kg/m <sup>3</sup> | float         |
| variance_noise                     | Variance Noise                                                         | dB                | float         |
| variance_respiratory_matrix_signal | Variance Respiratory Matrix Signal                                     | bpm               | float         |
| variance_temperature               | Variance Temperature                                                   |                   | float         |
| vdc                                | Vdc                                                                    | V                 | float         |
| vehicle_concentration              | Vehicle Concentration                                                  | car/m             | float         |
| vehicle_flow                       | Vehicle Flow                                                           | car/h             | float         |
| vehicle_occupancy                  | Vehicle Occupancy                                                      | -                 | float         |
| vehicle_speed_percentile           | Vehicle Speed Percentile                                               |                   | float         |
| vehicle_threshold_perc             | Vehicle Threshold Perc                                                 | %                 | float         |
| VehicleCapacity                    | weight capacity                                                        | t, Kg             | float         |
| VehicleRange                       | maxium capability of a vechicle                                        | t, Kg             | float         |
| velocity                           | Velocity                                                               | m/s, km/h         | float         |
| visibility                         | Visibility                                                             | -, Km             | integer,float |
| VOC                                | Volatile Organic Compounds                                             | ppm               | float         |
| VOC_2                              | Volatile Organic Compounds                                             | ppm               | float         |
| voc_3                              | Volatile Organic Compounds                                             | ppm               | float         |
| voltage                            | Voltage                                                                | V, millivolt, mV  | float         |
| wake_time_after_sleep_onset        | Wake Time After Sleep Onset                                            | min               | string        |
| wake_time_wrt_total_bed_time       | Wake Time Wrt Total Bed Time                                           | %                 | string        |
| waste_filling_rate                 | Waste Filling Rate                                                     | %                 | float         |
| water_consumption                  | Water Consumption                                                      | l/h               | float         |
| water_film                         | Water Film                                                             | µm                | float         |
| water_flowng                       | Water Flowing                                                          |                   | float         |
| water_level                        | Water Level                                                            | m                 | float         |
| weather                            | Weather                                                                | -                 | float,string  |
| weight                             | weight capacity                                                        | Kg                | float         |
| white_code_count                   | White Code Count                                                       | #                 | integer       |
| wifi_access_count                  | Wifi Access Count                                                      | #                 | integer       |
| wind                               | Wind                                                                   |                   | float         |
| wind_direction                     | Wind Direction                                                         | deg               | float         |
| wind_gust_speed                    | Wind Gust Speed                                                        | m/s               | float         |
| wind_speed                         | Wind Speed                                                             | m/s               | float         |
| yellow_code_count                  | Yellow Code Count                                                      | #                 | integer       |
| µg/L                               | TOC, Title of Concentration                                            | µg/Liter          | float         |