LIVING LAB

Be smart in a SNAP!

Smart City API
Web and Mobile App development

August 2020, Course
https://www.snap4city.org/577
scalable Smart aNalytic APplication builder for sentient Cities: for Living Lab and co-working with Stakeholders

https://www.Snap4City.org

Paolo Nesi, paolo.nesi@unifi.it
https://www.Km4City.org
https://www.disit.org

August 2020, Course
https://www.snap4city.org/577
Smart Cities

Industry 4.0

Integrated Smart Solutions

**Snap4 Framework**

- Expert System, Knowledge Base, Storage
- Big Data Analytics, Artificial Intelligence, Business Intelligence, Machine Learning, ...
- Data Flows, Workflows, MicroServices, Management
- Methodologies Courses and Community, Living Labs, Development tools, ...
- Dashboards and Apps, Control Rooms, Decision Support System, What-IF Analysis

Snap4City (C), August 2020
Main Organizations/areas

- Antwerp area (Be)
- Capelon (Sweden: Västerås, Eskilstuna, Karlstad)
- DISIT demo (multiple)
- Dubrovnik, Croatia
- Firenze area (I)
- Garda Lake area (I)
- Helsinki area (Fin)
- Livorno area (I)
- Lonato del Garda (I)
- Modena (I)
- Mostar, Bosnia-Herzegovina
- Pisa area (I)
- Pont du Gard, Occitane (Fr)
- Roma (I)
- Santiago de Compostela (S)
- Sardegna Region (I)
- SmartBed (multiple)
- Toscana Region (I), SM
- Valencia (S)
- Venezia area (I)
- WestGreece area (Gr)
Free Trial

• Register on [WWW.snap4city.org](http://WWW.snap4city.org)
  – Subscribe on DISIT Organization

• You can:
  – Access on basic Tools
  – Access to a large volume of Data
  – Create Dashboards
  – Create IOT Applications
  – Connect your IOT Devices
  – Exploit Tutorials and Demonstrations

IF you need to go more in deep you can ask us to pass at the next Role becoming full AreaManager with full rights of development, also for Data Analytics, machine learning, etc.
## On Line Training Material (free of charge)

<table>
<thead>
<tr>
<th>what</th>
<th>1st part (*)</th>
<th>2nd part (*)</th>
<th>3rd part (*)</th>
<th>4th part (*)</th>
<th>5th part (*)</th>
<th>6th part (*)</th>
<th>7th part (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF</td>
<td>General</td>
<td>Dashboards</td>
<td>IOT App, IOT Network</td>
<td>Data Analytics</td>
<td>Data Ingestion processes</td>
<td>System and Deploy Install</td>
<td>Smart City API: Web &amp; Mob. App</td>
</tr>
<tr>
<td>Interactive</td>
<td>Video1</td>
<td>Video2</td>
<td>Video3</td>
<td>Video4</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>duration</td>
<td>2:55</td>
<td>3:16</td>
<td>3:41</td>
<td>2:00</td>
<td>2:48</td>
<td>2:35</td>
<td>1:47</td>
</tr>
</tbody>
</table>

https://www.snap4city.org/577
General Overview of the full Course

- **1st part:** General Overview
- **2nd part:** Dashboards Creation and Management
- **3rd part:** IOT Applications development, IOT Devices, IOT Networks
- **4th part:** Data Analytics, in R Studio, in Python, how to Exploit and Manage Data Analytics in IOT Applications
- **5th part:** Data Ingestion, Data Warehouse, Data Gate, IOT Device Data ingestion, IOT App for Data Ingestion, etc.
- **6th part:** Snap4City Architecture, How To Install Snap4City
- **7th part:** Smart city API (internal and external) Web and Mobile App development tool kit

A number of the training sections include exercitations

Updated versions on: [https://www.snap4city.org/577](https://www.snap4city.org/577)

See also courses in ITALIANO: [https://www.snap4city.org/485](https://www.snap4city.org/485)
7th Part Agenda

• Smart City API: Internal and External
• Forging and Managing Flexible Mobile Apps, Web Apps and MicroApplications
  – Web and Mobile App with Open Development Kit
  – Understanding how City User are using the City Services
  – Engagement of City Users, towards a participated attitude
• Advanced Smart City API, MicroServices, Snap4City API
• Federated Knowledge Base and Smart City API
• Web and Mobile App Development Kit
• Acknowledgement
Snap4City (C), August 2020
Develop Mobile & Web Applications
Exploiting Snap4City Smart City Services

Analysis & Design
- Analysis
- Design
- Data Discovery
- Data Ingestion
- IOT App Development
- Data Analytics Development
- Dashboard Development
- Special Tool Development

Data Analytics
- IOT App Development
- Data Analytics Development
- Dashboard Development
- Special Tool Development

Smart City Services

Advanced Smart City API and MicroServices
- Snap4City Mobile & Web Apps
- Development Kit
- Application Requirements Analysis
- Application Development
- Deployment
- Testing
- Publication
- Production

Mobile and Web Apps
- Snap4City Mobile & Web Apps
- Development Kit
- Application Requirements Analysis
- Application Development
- Deployment
- Testing
- Publication
- Production
Levels of Difficulty

• Easy. non programmer level
  Some JavaScript rudiment coding
• Moderate. JavaScript programming
  Programming in R Studio
• Good. Exploiting Smart City API
  Developing Full IOT Applications, Dashboard and Mobile Apps
• Golden. Professional. Excellent.
Self Training main path

- Please start a fully guided training cases:
  - **HOW TO**: create a Dashboard in Snap4City
  - **HOW TO**: add a device to the Snap4City Platform
  - **HOW TO**: add data sources to the Snap4City Platform
  - **HOW TO**: define privacy rules for personal data, produced by the end-users own device
  - **HOW TO**: Develop Smart Applications, Snap4City development Life Cycle
  - **HOW TO**: HLT vs Ingestion, and HLT vs Widgets
  - **HOW TO**: Develop an IOT Application for Data Ingestion
  - **HOW TO**: Upload data into Knowledge Base, ServiceMap (triple upload)
  - **HOW TO**: Create as set of Devices with BulkProcessing
  - **HOW TO**: Create an IOT Device Model
  - **HOW TO**: Create an IOT Device Instance from IOT Directory tool
Smart City API: Internal and External
External Smart City API

https://www.km4city.org/swagger/external/index.html
Internal Snap4City API

https://www.km4city.org/swagger/internal/index.html
KM4City API

An exhaustive set of read-only APIs that have been developed in the context of the KM4City Project can be found below.

Service Discovery

Sample calls to APIs that allow to discover the available services, and retrieve some minimal information about each of them, including the Service URI, that can be leveraged for requesting further details through calls like those that can be found in the Details about services section (see below).

GET Search by category in a radius

https://servicemap.km4city.org/WebApp/63790/11.2280\maxDis=5&maxDis=36\categories=Accommodation\&lang=en

Search for an accommodation in a radius of 500 m from a given position, using English names & labels.

See also par. 4.2 of the Smart City API Guidelines.

PARAMS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>43.7809/11.2280</td>
</tr>
<tr>
<td>maxDis</td>
<td>Maximum distance from the given position of the services to be returned, expressed in kilometers. It defaults to 100 meters. If it is set to the special value &quot;inside&quot;, services are returned whose WKT boundary contains the given position (it could be the case of a point).</td>
</tr>
<tr>
<td>categories</td>
<td>Accommodation</td>
</tr>
</tbody>
</table>

300 – OK

```
{ "Services": [ { "Km4CityID": 15, "Type": "Accommodation", "Features": [ { "geometry": { "type": "Point", "coordinates": [ 11.2280, 43.7809 ] } } ] } ] }
```
• Advanced Smart City API which can be confined into a single Smart City installation or Federated as well as for Super Service Map
  – https://www.km4city.org/swagger/external/index.html
• Federated Multiple Snap4City Knowledge Bases. This allows the creation of mobile applications that may move from multiple cities and area accessing data and making queries transparently. This solution is presently in place among the Knowledge Bases of: Antwerp/Helsinki, Tuscany/Firenze, Sardegna, etc. The resulting Service is called Super Service Map and it is integrated in the Smart City API. For example, via:
  – https://www.disit.org/superservicemap/api/v1
• Federated Open Data Portals via DataGate/CKAN that presently presents now more than 13800 data sets linked for the cities of Helsinki and Antwerp.
  – https://datagate.snap4city.org/organization
  – Federation, Harvesting interface is: https://datagate.snap4city.org/harvest
• WFS service of Snap4City on top of Federated Smart City API or simple Smart City API of a single ServiceMap (smart City installation). This solution permits to GIS applications and platforms (such as ArcGIS OnLine ESRI, ArcGIS Enterprise ESRI, ArcGIS Map/pro Desktop, QGIS, GeoServer, etc.) to access at Snap4City data. For Example, via:
  – https://www.disit.org/superservicemap/api/v1/wfs
  – https://www.disit.org/superservicemap/api/v1/wfs?service=WFS&request=GetCapabilities&version=2.0.0
• WMS service of Snap4City for publishing maps and heatmaps, provided by an installed GeoServer third party open source tool. For example, via:
  – https://wmsserver.snap4city.org/geoserver/Snap4City/wms
  – https://www.km4city.org/swagger/external/index.html?urls.primaryName=Heatmap%20API
FORGING & MANAGING FLEXIBLE MOBILE APPS, Web Apps and MicroApplications
Web and Mobile App Developers, to generate Advanced Smart City API

Knowledge Base, Km4City

Km4City Open Source examples dev. tool kit

Mobile Apps

Web App HTML5

Embed into Web pages

City User

Mobile Application Monitoring Administrator

Swagger

DataInspector

ServiceMap

Snap4City (C), August 2020
Advanced SmartCity API

- Search data: by text, near, along, etc.
  - Resolving text to GPS and formal city nodes model
- Empowering city users: contributions, suggestions, forum discussions, etc.
- Events: Entertainment, critical and mobility
- Public and Private Mobility & Transport, and predictions
- POIs, Cultural and Touristic info
- Health services and predictions
- Environmental information, heatmaps; values
- Profiled Suggestions to City Users
- Traffic flow reconstruction
- Personal Assistant: PAVAL
- User Engagement: goal experiences, and assessment
- Sharing knowledge among cities ➔ see Knowledge base Management
Web and Mobile App with Open Development Kit
Km4City APP, features

- **5 languages:** IT, EN, SP, DE, FR
- **Profiles** city users: citizens, commuter, student, tourist, operator, etc.
- **Profiled Menu** per POI
  - adaptive
- **Main Menu:** dynamic, and personalized
- **Search Text**
- **Search per POI**
  - Near to you, near to a point, a line, ...
- **Other search**
  - Close to you, events green areas, public transport, tickets, Cycling, parking, ...
  - Etc.
- **POI**
  - Preferred, Social icon
  - Ranking, Comments, Images
Km4City APP

- Smart Parking, in Tuscany
- Smart First Aid in Tuscany
- Smart Public Transportation in Tuscany
- Smart Fuel pricing in Tuscany
- Bike Sharing in Pisa
- Weather condition in Tuscany
- Environmental data
- Pollution and Pollination in Tuscany
- Traffic Sensors in Tuscany
- Smart Routing in Tuscany
- Smart Transportation in Florence
  - Events, traffic, …
- Entertainment Events in Florence

Snap4City (C), August 2020
Km4City APP, features 3/3

• Navigation 3D
• Ticketing for busses
• App used are tool for city assessment
  – Wi-Fi status
  – iBeacon status
  – User behavior analysis
    • GPS movements kinds
    • OD matrix
    • International flows
Mobile App Features

- **Discovery** POI/services
- **Search**: POI, streets, suggestions
- **Mobility and transport**: Pub/priv, routing, car position, time table, park, sharing, tickets, etc.
- **Environment and Weather**: values, sensors, heatmaps, notifications
- **Assistant, Forum, Developer Assistant**
- **Goal Experiences** (Engagement)
- **Personal** data, activities, POI, tracking, IOT App, Dashboards, etc.
- **Events**: entertainment, critical
- **Sharing** position and trajectories with friends
- **Monitoring** city and personal Dashboards
- **Personalized for Operators and Developers** full control of their applications on cloud
ServiceMap Dev Tool (knowledge & Map tool)

- Search along a line
- Search around a GPS point

Smart City API call generation

- Web App HTML5
- Mobile Apps
- Embed into Web pages

http://www.disit.org/6873
Advanced Smart City API

• based on Km4City engine on the back office and much more
  – Documented: https://www.disit.org/6597

• **ServiceMap tool** is used to visually generate/request:
  – REST Calls to exploit the Smart City APIs in web and mobile applications. The examples of REST calls are sent by email.
  – views which can be embedded in web pages

• **Documentation:**
  – **TC5.15 - Snap4City Smart City API Collection and overview, real time**
  – **ServiceMap and ServiceMap3D, Knowledge Model, Km4City Ontology**
  – **Knowledge Base Graphs and Queries: browsing and queries into the KB**
Understanding how City Users are using the City Services
The App is a Bidirectional Device

- GPS Positions
- Selections on menus
- Views of POI
- Access to Dashboards
- searched information
- Routing
- Ranks, votes
- Comments
- Images
- Subscriptions to notifications
- ...

Produced information
- Accepted ?
- Performed ?
- ...

Derived information
- Trajectories
- Hot Places by click and by move
- Origin destination matrices
- Most interested topics
- Most interested POI
- Delegation and relationships
- Accesses to Dashboards
- Cumulated Scores from Actions
- Requested information
- Routing performed
- .....
Preferred Users’ Categories

- Public Transports: 34%
- Car park: 22%
- Cycle paths: 13%
- Around you: 17%
- Fuel station: 9%
- Gardens: 3%
- Citizen Menu: 1%
- Tourist Menu: 1%
User Behavior Analyser for Collective Profiling

Who

When

What

Where?

Why?

How move

Where they go ahead

Snap4City (C), August 2020
Scalable OD Matrix
Real Time Tracking: User Behaviour Analysis

How city users are moving! REAL TIME TRACKING
HeatMaps: Users as Sensors
User Behaviour Analyser

Hot places, trajectories, heatmap
Understanding City User Behaviour

- **Mobile Applications** can send data via Advanced Smart City API to collect data about the city usage by the city users via a signed consent
  - See Mobile and Web App: Toscana in a Snap, Helsinki in a Snap, Antwerp in a Snap.

- **City User behavior analysis** includes production of:
  - suggestions, trajectories, hot places/heatmaps, etc.
  - origin destination matrices
  - data for the city user engagement
  - Etc.

https://www.snap4city.org/drupal/node/489
Engaging City Users Towards Virtuous Participated Attitude
Profiled Engagements to City Users

• The users are profiled to learn habits:
  – Personal POI, paths, Mobility habits
• Information and engagements sent to the users are programmed according to the context and user behavior to:
  – Stimulate virtuous habits
  – More sustainable habits
  – More healthy habits, etc.
  – Get feedbacks
  – Provide bonus and prices, .....  
  – Send alerts, ....
Users’ Engagement

Inform
- Air Quality forecast is not very nice
- You have parked out of your residential parking zone
- The Road cleaning is this night
- The waste in S.Andreas Road is full

Engage
- Provide a comment, a score, etc.

Stimulate / recommend
- Events in the city, services you may be interested, etc..

Provide Bonus, rewards if needed
- you get a bonus since you parked here
- We suggest: leave the car out of the city, this bonus can be used to buy a bus ticket

Rules

User context

City context
Engaging City Users

• **Mobile Applications** can use Advanced Smart City API to collect data about the city usage by the city users via a signed consent.

• It can be used for sending engagements to them such as to:
  - **Inform**
    • You have parked out of your residential parking zone
    • The Road cleaning is this night
    • The waste in S.Andreas Road is full
  - **Engage**
    • Please Provide a comment, a score, etc.
  - **Stimulate / recommend**
    • Events in the city, services you may be interested, etc..
  - **Provide Bonus**
    • Since you have parked here you can get 1 Bonus
    • We suggest you to leave the car out of the city, this bonus can be used to buy a bus ticket
Engagement Manager

- Definition of Rules for campaigns
- Monitoring and follow-up for each City
- Segmented for user kind and interest
Sii smart. Sii-Mobility!
Scarica, viaggia, Vinci!

Dal 15 aprile al 15 luglio scegliere il trasporto pubblico ti premia!
Scarica l’app “Toscana dove, cosa”,
guadagna punti viaggiando in autobus e Vinci tanti fantastici premi!
Per maggiori informazioni visita il sito info.sii-mobility.org

Campaing on Sustainable Mobility

Dal 15 aprile al 15 luglio scegliere il trasporto pubblico ti premia!
Scarica l’app “Toscana dove, cosa”,
guadagna punti viaggiando in autobus e Vinci tanti fantastici premi.
Per maggiori informazioni visita il sito info.sii-mobility.org
Rules for Rewards

**ASSISTANCE**

• If public transport is detected after bus line suggestion on trajectory usually made on private transport → 10 points
  – Why don’t you take the bus line 4 in Piazza Marconi to reach your workplace? You save money, you respect the environment and you will be stress free for not worry about parking!

• Once a day, if public transport is detected after suggestion on an alternative bus line availability → 3 points
  – Why don’t you take the bus line 4 that stop just 50 meters far from you? You save money, you respect the environment and you will be stress free for the traffic jam!

• If public transport is detected for at least 30 minutes a day → 1 point

**ENGAGEMENT**

• Survey on commuter and their preferred way of mobility → 1 point
  – How many minutes you usually commute to go to work? How do you rate the service?

• Feedback on public transport → 1 point
  – Which current public transport are you using? Are the service in line with your expectation?

• Comments/Photo/Rate or survey on POI (public transport) → 1 point

• Survey on use of the App after N days or for tourist coming home → 1 point

• Feedback on PPOI or mobility → 1 point
Current Numbers

From 1° September 2016

– Detected 2108 PPOIs on 1080 users
  • 437 HOME
  • 285 WORK
  • 34 SCHOOL
  • 1350 EXTRA

– 130 PPOIs are feedbacked
– 460 survey responses

From 1° August 2017

– Built 524 Markov Networks about user’s trajectories
## Validation of user Engagement

<table>
<thead>
<tr>
<th>Months</th>
<th>Msg Sent</th>
<th>Msg Viewed</th>
<th>Msg Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-January</td>
<td>3888</td>
<td>380</td>
<td>12</td>
</tr>
<tr>
<td>2-February</td>
<td>4319</td>
<td>492</td>
<td>22</td>
</tr>
<tr>
<td>3-March</td>
<td>4739</td>
<td>450</td>
<td>25</td>
</tr>
<tr>
<td>4-April</td>
<td>6567</td>
<td>918</td>
<td>67</td>
</tr>
<tr>
<td>5-May</td>
<td>7594</td>
<td>972</td>
<td>61</td>
</tr>
<tr>
<td>6-June</td>
<td>6437</td>
<td>695</td>
<td>55</td>
</tr>
<tr>
<td>7-July</td>
<td>9432</td>
<td>697</td>
<td>69</td>
</tr>
<tr>
<td>8-August</td>
<td>6988</td>
<td>429</td>
<td>73</td>
</tr>
<tr>
<td>9-September</td>
<td>5885</td>
<td>345</td>
<td>49</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>55849</strong></td>
<td><strong>5375</strong></td>
<td><strong>433</strong></td>
</tr>
</tbody>
</table>

---

![Graph showing user engagement metrics](image)

**Note:**

Snap4City (C), August 2020
User Behaviour Analysis

VALIDATION

• During the PILOT new rules has been added (30 on a total of 80) and mostly all of them are still online

• COMMENT_POI: requires more user interaction and not very contextualized (POI proximity) → higher rate of sent, lower rate on execution

• MOBILITY_FEEDBACK: requires less user iteration and very contextualized (user in MOBILITY) → normal rate of sent, high rate on execution

<table>
<thead>
<tr>
<th></th>
<th>Msg Sent</th>
<th>Msg Viewed</th>
<th>Msg Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT_POI</td>
<td>21632</td>
<td>804</td>
<td>15</td>
</tr>
<tr>
<td>MOBILITY_FEEDBACK</td>
<td>5378</td>
<td>371</td>
<td>94</td>
</tr>
</tbody>
</table>
Dashboard created to monitor in real time the answers to the survey provided on the Mobile App directly by the Engagement tool.

Dashboard monitoring the Mobile App:

- Collecting the clicks
- Describing the community of users in terms of the profile aspects
- Measuring the time spend, and topics of interest of the users, etc.
Dashboard created to monitor in real time the answers to the survey provided on the Mobile App directly by the Engagement tool.
Dashboard monitoring the Mobile App:

- Collecting the clicks
- Describing the community of users in terms of the profile aspects
- Measuring the time spend, and topics of interest of the users, etc.
Connected Drive
Main Concept of Connected Drive

- Different kinds of communications may arrive on the vehicles on board devices
- Mobile Phones can be a possible facilities

- Geolocated Real time Information as:
  - Alerting, dynamic digital signage (may not present physically on the road)
  - Supporting autonomous driving vehicles
Experimentation on «Toscana Dove Cosa»

• Mobile App supporting connected Drive V2I connections:
  – For the MOSAIC project and pilot in Tuscany

• The mobile App has a Navigator which includes now the acquisition of connected drive messages
Scenario
ADVANCED SMART CITY API, MICROSERVICES, SNAP4CITY API
Some structures from Km4City model

ServiceMap: [https://servicemap.km4city.org](https://servicemap.km4city.org)
Advanced Development Kit features

• Exploiting Km4City Advanced Smart City API
  – Open Source: GitHub
  – Multiplatform: exploiting Apache Cordova Framework
  – Active since 2015
  – Adopted by a community of several Projects, Cities and SME

• Respecting user privacy:
  – Anonymous usage vs Authenticated usage (OAuth, email, ...)

• Modular & Dynamic:
  – Loading new modules from the WEB, and/or creating App by modular approach

• Personalization and Profiling:
  – Personalized menu, proposed POI for search

• Reaching City Users:
  – Alerting and notifications by location, by user behaviour
Advanced SmartCity API

- Search data: by text, near, along, etc.
  - Resolving text to GPS and formal city nodes model
- Empowering city users: contributions, suggestions, forum discussions, etc.
- Events: Entertainment, critical and mobility
- Public and Private Mobility & Transport, and predictions
- POIs, Cultural and Touristic info
- Health services and predictions
- Environmental information, heatmaps; values
- Profiled Suggestions to City Users
- Traffic flow reconstruction
- Personal Assistant: PAVAL
- User Engagement: goal experiences, and assessment
- Sharing knowledge among cities → see Knowledge base Management

Swagger
Advanced Smart City API

https://serviceMapper.dist.it/WebAppAPI/3pizza/1

SMART CITY API WEB DOCUMENTATION

Servers

https://servicemap.dist.it/WebAppAPI/3pizza/1

Services

GET / Service discovery and information

- **Service search near GPS position**: It allows to retrieve the list of services that are near a given GPS position. The services can be filtered as belonging to specific categories (e.g., Accommodation, Hotel, Restaurant, etc.) or having specific words in any textual field. It can also be used to find services that have a WKT spatial description that contains a specific GPS position.
- **Service search near a service**: It allows to retrieve the list of services that are near a given service identified by its serviceId. The services can be filtered as belonging to specific categories (e.g., Accommodation, Hotel, Restaurant, etc.) or having specific words in any textual field. It can also be used to find services that have a WKT spatial description that contains a specific GPS position.
- **Service search within a WKT area**: It allows to retrieve the list of services that are inside a rectangular area. The services can be filtered as belonging to specific categories (e.g., Accommodation, Hotel, Restaurant, etc.), or having specific words in any textual field.
- **Service search within a WKT described area**: It allows to retrieve the set of services that are inside a geographic region described using the Well-Known Text (WKT) format. The services can be filtered as belonging to specific categories (e.g., Accommodation, Hotel, Restaurant, etc.) or having specific words in any textual field.
- **Service search within a stored WKT described area**: It allows to retrieve the set of services that are inside a geographic region described using the Well-Known Text (WKT) format, by referring to the WKT with an identifier provided when the WKT is stored. The services can be filtered as belonging to specific categories (e.g., Accommodation, Hotel, Restaurant, etc.), or having specific words in any textual field. The list of available geometries can be retrieved from the **Service Map** in the Search Area selection box (with Search Range specific area). New geometries can be provided using the [http://www.kml胸怀.com/p/L](http://www.kml胸怀.com/p/L) web service which can store a WKT from a string or by providing directly the WKT string.
- **Service search by municipality**: It allows to retrieve the set of services that are in a specific municipality. The services can be filtered as belonging to specific categories (e.g., Accommodation, hotel, Restaurant, etc.), or having specific words in any textual field.
- **Service search by query ID**: It allows to retrieve the set of services associated with a query stored using the **Service Map** user interface.
- **Full text search**: It allows to retrieve the geospatial entities (not only services) that match with a list of keywords. The results can be possibly filtered to be within a specified distance from a GPS position, or within a rectangular area or inside a WKT geospatial area.
- **Service info**: It allows to retrieve information about a service using its serviceId, as an HTML format query parameter set to json or a machine readable JSON document format (query parameter set to json).

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>Through this parameter, the user indicates where the services have to be searched. It could be a boundary within which to search, or a point around which to search.</td>
</tr>
<tr>
<td>str</td>
<td>/ (query)</td>
</tr>
</tbody>
</table>

Usage & Sample values:
Thematic Data Domain Tuscany

- Street and geoinformation of the territory and details for routing, navigation, ...
- GeoResolution, Environmental data
- Mobility and Transport: public and private, public transport, parking status, fuel stations prices, traffic sensors, etc.
- Culture and Tourism: POI, churches, museum, schools, university, theatres, events in Florence
- Environmental: pollution real time, weather forecast, etc.
  - Environmental data geo resolution
- Social Media: twitter data
- Health: hospital, pharmacies, status of the first aid triage in major hospitals, ...
- Alarms: civil protection alerts, hot areas, ...

Snap4City (C), August 2020
Access to Point of Interest information, POI

- **POI**: point of interest
- **type**: macro and subcategories
- **Position**: GPS, address, telephone, fax, email, URL, ...
- **Description**: textual, multilingual, with images, ...
- **Link** to dbPedia, Linked Open Data
- **Links to other services**
- **Real time data if any**: sensors data, timeline, events, prices, opening time, rules of access, status of services, status of queue, etc..

- *See transversal services on ServiceMap – Regular and in test platform*
Concepts of Services: Macro and subcathegory

A SKOS area into the Km4Clty Ontology and Knowledge base for modeling POI and any element on map
Service Information: different kinds of services
General Text Search Features

Search by text for POIs via:

- Full text: description, title, macro and category name
- Filtering by macro-cat and subcategory
- Filtering on distance and geometric shape

Search by text with assisted suggestion to get:

- Streets and civic numbers, or POI, locations

Geo resolution, from point to street; from civic to GPS, etc.
Search by Shape and Distance

Each request or search in the Km4City model can be referred to a point and a ray, to an area, to a polyline.

- Around a point or POI
- Inside a closed polyline
- Along a polyline

Inside an area
Empowering City Users

• Allow city users to
  – provide comments, images and scores associated with a certain Service (or place, via GPS), discussions on forums, etc.
  – Get list of last contributions of the same kind provided by other users
  – Save favorites
  – Share trajectories,
  – Save and Manage their own data, IOT data, etc.

• Contributions can be:
  – used as feedbacks
  – moderated by a back-office personnel
  – ...

• Connection with powerful servers based on 311 standard it also possible
Access to Event information

• Getting Traffic Events: ESB, etc.
• Getting Critical Events: CAP standard
• Getting Police events
• Getting Entertainment Events in the city
  – Theater, museum, show, sport, etc.
  – Getting Event details
    • Event kind, and thus ordering
    • in the day, week, and month
    • Location, and thus ordering, or selecting events per area, per residence
    • General information
    • Opening and cost (if any)
    • Etc.
Supporting City Users in using Public Mobility

Public Transportation, PT

– Getting tickets
– Getting bus stops, lines, and timelines for bus, train and tramline (GTFS, ETL, ...)
– Getting Tunnel and Ferry Status
– Searching Services along a Pub. Transport line or closer to a stop
– Searching the closest bus stops
– searching for BUS stops via name
– real time delays of busses
– Modal/multimodal routing for Pub. Transport
– Tracking fleets, trajectories, etc.
– Get connected drive data
Supporting City Users using Private Mobility

Private Transport

– Parking status (DATEX II, ...)  
– Saving car park  
– Getting closer parking  
– OBD2 data from your engine or fleet  
– Getting parking forecast: short and long term  
– Getting closer free space on parking  
– Getting fuel stations location and fuel product prices  
– Getting bike sharing rack status  
– Searching Services along a path or closer to a point or Service as Hotel, Restaurants, square, etc.  
– Getting closer cycling paths  
– Recharging stations: location and status  
– Getting traffic information  
– Heatmap where is safer to bike
Private Mobility: routing and navigation paths

To get the path from two points/POIs:
- Shortest for pedestrian
- Quietest for pedestrian
- Shortest for private vehicles
- Multimodal with Public Transportation
- Constrained routing

Search for POIs along the identified Path!

http://www.disit.org/ServiceMap
New Experience to access at Cultural and Touristic info

• Getting location and description of Point of Interests, POIs: culture and tourism first
  – Location, images, phone, URL, etc.
  – Get image, video, audio, ...

• Search for POIs in areas and closer

• Get routing to reach location or POI by walking downtown
  – searching Services along the path

• Search for location, full text assisted

• Leave a score, take a picture, etc.
New way to access at health services

• Searching for pharmacies and hospitals
• Getting the closest hospital first aid locations and status
• Getting real time updated information about the first aid status of major hospitals (triage)
Access at Environmental information

• Getting weather forecast for the next hours and days
• Getting alert information from Civil protection
• Getting air quality status
• Getting Air quality via heatmaps, heatmap animation
• Computing Air quality indexes
• Computing Air quality predictions
• Getting pollination status
• Getting actual weather status: temperature, humidity, pressure, rain level,
  etc.
Profiled Suggestions to City Users

Personalized suggestions

– The server provide suggestions in the user context (location and time) arranged in a number of categories
  • Culture, mobility, food and drink, etc.
  • Alerts: civil protection, city council, twitter data, etc.

– The city user may reject some of them, thus the suggestion engine learns about preferred topics and category
Profiled Engagements to City Users

• The user are profiled to learn habits:
  – Personal POI and paths
  – Mobility habits

• Information and engagements sent to the city users are programmed according to the user evolution to:
  – Stimulate virtuous habits
  – More sustainable habits
  – More healthy habits, etc.
  – Get feedbacks
  – Provide bonus and prices, ...
  – Send alerts, ...
PAVAL: Personal Assistant

• Your Personal Assistant for navigating in the city
• Ask PAVAL to get help and information about the city services
• ITA, ENG
• Active on Florence and whole Tuscany
• Mobile and PC

https://assistant.disit.org
Developing Web and Mobile Apps, MicroApps...

Knowledge Base, Km4City

Snap/Km4City Open Source development tool kit

Swagger

Advanced Smart City API

Mobile Apps

Web App HTML5, MicroApplications

Embed into Web pages

City User

Developer

Mobile Application Monitoring Administrator

ServiceMap

DataInspector

Snap4City (C), August 2020
ServiceMap Dev Tool (knowledge & Map tool)

- Search around a GPS point
- Search along a line
- Smart City API call generation
- Web App HTML5
- Mobile Apps
- Embed into Web pages

http://www.disit.org/6873
Smart City API

• based on Km4City engine on the back
• documented: https://www.snap4city.org/404
• ServiceMap tool to generate visually calls to exploit the Smart City API in web and mobile applications
• Documentation and examples:
  – TC5.15 - Snap4City Smart City API Collection and overview, real time
  – ServiceMap and ServiceMap3D, Knowledge Model, Km4City Ontology
  – Knowledge Base Graphs and Queries: browsing and queries into the KB
• The Alternatives:
  – just Dashboards directly exploiting data on graphics and/or
  – IOT Applications via Node-RED exploiting MicroServices also using the Smart City APIs
Federated Knowledge Bases and Smart City APIs
Km4City Federation

At different levels:
- Among cities/regions
- Among data providers
- Among Operators

By Means of:
- Smart City API → Apps
- Km4City Smart City Ontology
- Dashboards/data analytics
Multiple Knowledge Bases Km4City/ServiceMap

- A Mobile App may refer to one Smart City API Server (for Area 1) via SUPER and receive data from the Federated SUPERs (Area 2) if navigation, queries, etc. are leading to discover out of the addressed KB.
  - SUPER can be used for creating redundant and/or balanced distributed solutions for Federated KB. See Area 2, the two KB in the front.
  - Federated SUPER ServiceMap can have overlapped KB even totally.
  - A Mobile App can be developed to support multiple Smart City API servers, for balancing and
- The usage of Super (ServiceMap) is not mandatory so that separate services can be produced as well
- SuperServiceMap and ServiceMap presents the same Smart City APIs.
Super Service Map

- ServiceMap do not need to permanently share data
- Distributed Searches are performed with $o(1)$ complexity
- Each ServiceMap can be of any size
- Each ServiceMap may have different number of services
- Services of SCAPI in ServiceMap Can be georeferenced or not
- Clients can pass from one SuperServiceMap to another transparently: moving devices
- The network of ServiceMap can be reconfigured dynamically
- Results from a API rest call are provided in real time also when the size of the network is large
Federated ServiceMap and Smart City API

To improve scalability, fault tolerance and federation among cities:

– One entry point Smart City API for all zones
– Multiple Knowledge base See performance assessment

At different levels:

– Among cities/regions
– Among data providers, Operators

By Means of:

– Smart City API → Apps
– Smart City Ontology
– Dashboards/data analytics
– Organization independent
Exposing Services

- **Advanced Smart City API** which can be confined into a single Smart City installation or Federated as well as for Super Service Map
  - [https://www.km4city.org/swagger/external/index.html](https://www.km4city.org/swagger/external/index.html)
- **Federated Multiple Snap4City** Knowledge Bases. This allows the creation of mobile applications that may move from multiple cities and area accessing data and making queries transparently. This solution is presently in place among the Knowledge Bases of: Antwerp/Helsinki, Tuscany/Firenze, Sardegna, etc. The resulting Service is called Super Service Map and it is integrated in the Smart City API. For example, via:
  - [https://www.disit.org/superservicemap/api/v1](https://www.disit.org/superservicemap/api/v1)
- **Federated Open Data Portals** via DataGate/CKAN that presently presents now more than 13800 data sets linked for the cities of Helsinki and Antwerp.
  - [https://datagate.snap4city.org/organization](https://datagate.snap4city.org/organization)
  - Federation, Harvesting interface is: [https://datagate.snap4city.org/harvest](https://datagate.snap4city.org/harvest)
- **WFS service of Snap4City** on top of Federated Smart City API or simple Smart City API of a single ServiceMap (smart City installation). This solution permits to GIS applications and platforms (such as ArcGIS OnLine ESRI, ArcGIS Enterprise ESRI, ArcGIS Map/pro Desktop, QGIS, GeoServer, etc.) to access at Snap4City data. For Example, via:
  - [https://www.disit.org/superservicemap/api/v1/wfs](https://www.disit.org/superservicemap/api/v1/wfs)
  - [https://www.disit.org/superservicemap/api/v1/wfs?service=WFS&request=GetCapabilities&version=2.0.0](https://www.disit.org/superservicemap/api/v1/wfs?service=WFS&request=GetCapabilities&version=2.0.0)
- **WMS service of Snap4City** for publishing maps and heatmaps, provided by an installed GeoServer third party open source tool. For example, via:
  - [https://wmsserver.snap4city.org/geoserver/Snap4City/wms](https://wmsserver.snap4city.org/geoserver/Snap4City/wms)
  - [https://www.km4city.org/swagger/external/index.html?urls.primaryName=Heatmap%20API](https://www.km4city.org/swagger/external/index.html?urls.primaryName=Heatmap%20API)
Web and Mobile App Development Kit
• **Apache Cordova** is a set of **JavaScript APIs** that enable the devices to the application developer to access native features of the device such as the camera or accelerometer, storage, network, gps ....

• Combined with a user **interface framework** such as Dojo Mobile or jQuery Mobile or Sencha Touch, allows the development of smartphone applications using only **HTML, CSS and JavaScript**.

• When using the Cordova API, an application can be built without any native code (Java, Objective-C, C# etc.). The **web technologies** used are **hosted in the same application** at the local level (usually not on a remote http server).

• These **JavaScript API** are **consistent** and **valid** for the **different platforms** of mobile devices, in this way the application built on the Web standard, should be **portable** with a **minimum of changes**.
Mustache JS

• The library is **independent** from specific framework but there are plugins for the integration with jQuery, Dojo, and YUI.

• Possibility to work with **javascript objects** and then exploit the communication of data in **JSON format from a REST** call via AJAX.

• The **templates** for Mustache may be assigned or loaded as a string to a variable and the placeholder are identified by two braces, for example: `{{miopplaceholder}}`.

• One of the most interesting of the library feature is support in **enumerable values**

• Documentation and downloads are available on the official website: [http://mustache.github.io](http://mustache.github.io)
var data = {
  risultato: true,
  titolo: Città italiane,
  descrizione: Lista delle città italiane,
  citta: [
    {nome: Milano, sigla: MI},
    {nome: Roma, sigla: RM}
  ]
};

<h1>{{titolo}}</h1>
<p>{{descrizione}}</p>
{{#risultato}} //solo se risultato è true
  <ul>{{#citta}}
    <li>{{nome}} ({{sigla}})</li>
  {{/citta}}</ul>
{{/risultato}}
{{^risultato}} //altrimenti...
  <p><em>Nessuna città trovata!</em></p>
{{/risultato}}
Mustache JS

```html
<h1>{{titolo}}</h1>
<p>{{descrizione}}</p>
{#{risultato}} // solo se risultato è true
  <ul>{#{citta}}
    <li>{{nome}} ({{sigla}})</li>
  </ul>
{#{citta}}
{/risultato}
{#{^risultato}} // altrimenti...
  <p><em>Nessuna città trovata!</em></p>
{/risultato}
```

Template + JSON + Mustache

**Città italiane**

- Milano (MI)
- Roma (RM)
OpenLayers 3.0

- **OpenLayers** is an open source JavaScript library for displaying map data in web browsers and can be used with a hybrid application developed with Cordova.

- In the **early versions** of the app, the map was managed by **Leaflet.js** library. This was replaced because it didn't support the rotation, which is required to insert navigation functions within the app.

- In addition, OpenLayers 3.0 builds the map and objects added to it with a **canvas** renderer, which is **very efficient** when objects are **numerous and small** as the markers displayed for each search done with the app.

- Documentation and downloads are available on the official website: [http://openlayers.org](http://openlayers.org)
General architecture of Mobile / Web App

Mobile App Snap4City and/or Km4City

User Interface
- Suggestions
- View Map
- Map Navigation
- Settings
- Info
- Crowd
- Search
- paths
- Bonus
- Suggestions on Time
- Easy Parking
- How move?
- ehealth
- What Move?
- What think?
- Do you know?
- It could be...?
- Authentication

Smart City API, Conditional Access System

Support of Integrated Interoperability

Knowledge Base

Algorithms

IOT App / ETL: data ingestion

Big Data processing grid

Internal and external data algorithms / processes

User Profiler & Suggestion on demand

User Crowd Sourcing

Manager & Engager

Algorithm Loader: IOT App, ETL and/or Java

Module based
Data sources
Data provided

WebKit

Plug Interface in JavaScript

Android Plugin Implementation

iOS Plugin Implementation

Windows Phone Plugin Implementation

HW .....
Create ParkingSearcher Module

In the slides following there is an example of how to add a module to the app.

The goal of this example is to create a new module that in addition to viewing the list of car parks as is already the case for the button named “Parking" will show directly the number of free parking lots for each car park found.
Create ParkingSearcher Module

- Files required for creating a new module are as follows

  - A Javascript file containing the logic
  - 5 JSON files (ita, eng, esp, deu, fra) containing labels to be included in the new interface
  - A JSON file that contains one or more buttons to be added to principal menu to allow the user to interact with the newly created module
Create ParkingSearcher Module

- Copy these files to a **new folder** that will have the **name of the new module** (i.e., ParkingSearcher): the **names of the files** copied have to be changed to get the **module name as a prefix**

```
oroworkspacesiiMobilityAppKitwwwjsmodulesparkingSearcher
```

- `ParkingSearcher.js`
  - Tipo: File JavaScript
- `parkingSearcher.labels.deu.json`
  - Tipo: JSON File
- `parkingSearcher.labels.eng.json`
  - Tipo: JSON File
- `parkingSearcher.labels.fra.json`
  - Tipo: JSON File
- `parkingSearcher.labels.ita.json`
  - Tipo: JSON File
- `parkingSearcher.labels.spa.json`
  - Tipo: JSON File
- `parkingSearcher.principalMenu.json`
  - Tipo: JSON File
ParkingSearcher in main menu

• Field descriptions for creating buttons in the main menu

This field contains the callback for the new module.

The present callbacks should be left, because they serve to close the main menu and to center the map on the GPS.
ParkingSearcher in main menu

- **Field descriptions** for creating buttons in the **main menu**

```javascript
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#000000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "textColor": "#000000",
  "captionId": "principalMenuParkingSearcher",
  "captionTextId": "moduleParkingSearcher",
  "step": true,
  "stepId": "eventsBadge",
  "ribbon": "true",
  "ribbonId": "",
  "ribbonStyle": "background: #336633; background: linear-gradient(#33FF33 0%, #336633 100%);"
}
```

These blocks of fields are **mutually exclusive**. Allow you to choose the icon that will identify the button that you are creating. This icon can be chosen as an **image**, a **text**, a **glyphicon** (Bootstrap) or **ionicons** (ionicons.com).

N.B. Field **iconId** can be useful if you plan to edit the selected icon **dynamically**.
ParkingSearcher in main menu

- **Field descriptions** for creating buttons in the main menu

These blocks of fields are **mutually exclusive**. Allow you to choose the icon that will identify the button that you are creating. This icon can be chosen as an image, a **text**, a **glyphicon** (Bootstrap) or **ionicons** (ionicons.com).

N.B. Field **iconId** can be useful if you plan to edit the selected icon **dynamically**.

```json
{
   "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
   "iconId": "",
   "iconClass": "icon ion-android-bus",
   "iconFontSize": "41px",
   "iconColor": "#CC0000",
   "imgSrc": "img/ticketmenu.png",
   "imgHeight": "37px",
   "text": "P",
   "textFontSize": "39px",
   "textColor": "#CC0000",
   "captionId": "moduleParkingSearcher",
   "captionTextId": "moduleParkingSearcher",
   "step": true,
   "stepId": "eventsBadge",
   "ribbon": true,
   "ribbonId": "",
   "ribbonStyle": "background: #336633; background: linear-gradient(90deg, #336633 0%, #666633 100%);"
}
```

parkingSearcher.principalMenu.json
ParkingSearcher in main menu

• Field descriptions for creating buttons in the main menu

```json
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMap();"
  "icon": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#C00000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "39px",
  "textContent": "#C00000",
  "captionId": "principalMenuParkingSearcher",
  "captionTextId": "moduleParkingSearcher",
  "step": false,
  "stepId": "eventsBadge",
  "ribbon": "true",
  "ribbonId": "",
  "ribbonStyle": "background: #336633; background: linear-gradient(#33FF33 0%, #336633 100%);"
  "ribbonText": "Beta",
  "removed": false,
  "index": 0
}
```

captionId serves to indicate the container tag of the text that is located at the bottom of each button.

captionTextId indicates the name of the field in labels.*.json whose value is the text to be inserted in the previous container.

parkingSearcher.principalMenu.json
ParkingSearcher in main menu

• Field descriptions for creating buttons in the main menu

These blocks of fields are used to show the user badges containing information related to the button on which are located
• Field descriptions for creating buttons in the main menu

These blocks of fields are used to show the user badges containing information related to the button on which are located
ParkingSearcher in main menu

- **Field descriptions** for creating buttons in the **main menu**

```json
{
    "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
    "icon": "",
    "iconClass": "icon icon-android-bus",
    "iconFontsize": "41px",
    "iconColor": "#cc0000",
    "imgSrc": "img/ticketmenu.png",
    "imgHeight": "37px",
    "text": "P",
    "textFontsize": "39px",
    "textColor": "#cc0000",
    "captionId": "principalMenuParkingSearcher",
    "captionTextId": "moduleParkingSearcher",
    "step": true,
    "stepId": "eventsBadge",
    "ribbon": true,
    "ribbonId": "",
    "ribbonStyle": "background: #336633; background: linear-gradient(#33FF33 0%, #336633 100%);"
    "ribbonText": "Beta",
    "removed": false,
    "index": 0
}
```

**removed** field is useful to allow the removal and the insertion of the buttons in the main menu by the user.

**index** field is useful for rendering the buttons in the order chosen by the user.

parkingSearcher.principalMenu.json
ParkingSearcher in main menu

• Field descriptions for creating buttons in the main menu
ParkingSearcher in main menu

- Loading **new buttons modules** within the main menu, takes place by comparing the **captionId** field.

- If the menu already has a button with the **same captionId**, the first is **replaced** with the **new one**.

- To **remove** a button from the main menu (field **removed** hides it) add a **delete** field with value equal to **true**.
ParkingSearcher in main menu

- First version of the button

```json
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "",
  "iconFontSize": "",
  "iconColor": "",
  "imgSrc": "",
  "imgHeight": "",
  "text": "LP",
  "textFontSize": "38px",
  "textColor": "#CC0000",
  "captionId": "principalMenuParkingSearcher",
  "captionTextId": "moduleParkingSearcher",
  "step": "",
  "stepId": "",
  "ribbon": true,
  "ribbonId": "",
  "ribbonStyle": "background: #CC0000;background: linear-gradient(#FF6600 0%, #CC0000 100%);",
  "ribbonText": "NEW",
  "removed": false,
  "index": 0
}
```

parkingSearcher.principalMenu.json

Label missing
Labels of ParkingSearcher

• Description of label.*.json files

Three important things to check:

• Languages shall be indicated by 3 characters: ita, deu, esp, fra, eng
• The label for the button must be contained within the object "principalMenu"
• The name of the field inside "principalMenu“ must be the same of “captionTextId” seen before
Labels of ParkingSearcher

- Description of label.*.json files

```javascript
$(captionId).html(labels.principalMenu[captionTextId]);
```

parkingSearcher.principalMenu.json

```
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": ",",
  "iconClass": ",",
  "iconFontSize": ",",
  "iconColor": ",",
  "imgSrc": ",",
  "imgHeight": ",",
  "text": "LP",
  "textFontSize": "38px",
  "textColor": ">

```

label.ita.json

```

```
{
  "principalMenu": {
    "moduleParkingSearcher": "Lista Parcheggi"
  }
}
```

label.eng.json

```
{
  "principalMenu": {
    "moduleParkingSearcher": "Car Park List"
  }
}
```

label.deu.json

```
{
  "principalMenu": {
    "moduleParkingSearcher": "Parkplatz Liste"
  }
}
```

label.fra.json

```
{
  "principalMenu": {
    "moduleParkingSearcher": "liste parkings"
  }
}
```

label.esp.json
Labels of ParkingSearcher

• Description of label.*.json files

<table>
<thead>
<tr>
<th>Language</th>
<th>Label Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ita</td>
<td>Lista Parcheggi</td>
</tr>
<tr>
<td>eng</td>
<td>Car Park List</td>
</tr>
<tr>
<td>deu</td>
<td>Parkplatz Liste</td>
</tr>
<tr>
<td>fra</td>
<td>Liste parkings</td>
</tr>
<tr>
<td>esp</td>
<td>Lista de Aparcamiento</td>
</tr>
</tbody>
</table>

parkingSearcher.principalMenu.json
Create ParkingSearcher Module

- It is seen as fill most of the files in the folder of new module ParkingSearcher that is developed in this presentation

```
oro > workspace > siiMobilityAppKit > www > js > modules > parkingSearcher
```

- ParkingSearcher.js: File JavaScript
- parkingSearcher.labels.deu.json: JSON File
- parkingSearcher.labels.eng.json: JSON File
- parkingSearcher.labels.fra.json: JSON File
- parkingSearcher.labels.ita.json: JSON File
- parkingSearcher.labels.spa.json: JSON File
- parkingSearcher.principalMenu.json: JSON File

TODO

✔ ✔ ✔ ✔ ✔ ✔
ParkingSearcher Module Functions

- Functions contained in ParkingSearcher.js

```javascript
show: function () {
  application.resetInterface();
  MapManager.showMenuReduceMap("#" + ParkingSearcher.idMenu);
  
  $("#" + ParkingSearcher.idMenu + "Collapse").hide();
  ParkingSearcher.open = true;
  InfoManager.addingMenuToManage(ParkingSearcher.varName);
  application.addingMenuToCheck(ParkingSearcher.varName);
  application.setBackButtonListener();
},

hide: function () {
  
  $("#" + ParkingSearcher.idMenu).css({ 'z-index': '1001' });
  MapManager.reduceMenuShowMap("#" + ParkingSearcher.idMenu);
  InfoManager.removingMenuToManage(ParkingSearcher.varName);
  application.removingMenuToCheck(ParkingSearcher.varName);
  ParkingSearcher.open = false;
},
```

Closes any previously opened menu, shrinks the map to display the menu, hides the button to reduce the menu, since it will open already reduced.

Recording to other variables to get notifications when:
- users press the back button
- users change the device orientation
- must be closed the menu opened by this module
ParkingSearcher Module Functions

• Functions contained in ParkingSearcher.js

```javascript
show: function () {
  application.resetInterface();
  MapManager.showMenuReduceMap("#" + ParkingSearcher.idMenu);
  $("#" + ParkingSearcher.idMenu + "Collapse").hide();
  ParkingSearcher.open = true;
  InfoManager.addingMenuToManage(ParkingSearcher.varName);
  application.addingMenuToCheck(ParkingSearcher.varName);
  application.setBackButtonListener();
},

hide: function () {
  $("#" + ParkingSearcher.idMenu).css({'z-index': '1001'});
  MapManager.reduceMenuShowMap("#" + ParkingSearcher.idMenu);
  InfoManager.removingMenuToManage(ParkingSearcher.varName);
  application.removingMenuToCheck(ParkingSearcher.varName);
  ParkingSearcher.open = false;
},
```

Does the opposite functions to those performed by the function show, also reset the z-index of the menu.
ParkingSearcher Module Functions

- **Functions** contained in ParkingSearcher.js

```javascript
checkForBackButton: function () {
  if (ParkingSearcher.open) {
    ParkingSearcher.hide();
  }
},
refreshMenuPosition: function () {
  if (ParkingSearcher.open) {
    MapManager.showMenuReduceMap("#" + ParkingSearcher.idMenu);
    Utility.checkAxisToDrag("#" + ParkingSearcher.idMenu); 
    if (ParkingSearcher.expanded) {
      ParkingSearcher.expandBusRoutesMenu();
    }
  }
},
closeAll: function () {
  if (ParkingSearcher.open) {
    ParkingSearcher.hide();
  }
},
```

These are the **callbacks** called to **notify** the occurrence of an event among those described previously (see show function) and for which we recorded the module:

- users press the **back button**
- users change the **device orientation**
- must be **closed the menu** opened by this module
ParkingSearcher Module Functions

- Functions contained in ParkingSearcher.js

```javascript
refreshMenu: function () {
    if ($("#" + ParkingSearcher.idMenu).length == 0) {
        $('.indexPage').append("<div id="" + ParkingSearcher.idMenu + "" class="commonHalfMenu"></div>")
        ViewManager.render(ParkingSearcher.results, "#" + ParkingSearcher.idMenu, "ParkingMenu");
        Utility.movingPanelWithTouch("#" + ParkingSearcher.idMenu + "ExpandHandler", "#" + ParkingSearcher.idMenu);
    } else {
        // More code here...
    }
}
```

- Checks if there is the element that will contain the html code created through the use of Mustache library.
- It is generated the html code with template ParkingMenu.mst.html and JSON ParkingSearcher.results and added to the element container.
- Finally, the feature that allows the users to widen the menu by dragging the handler is added to it.
ParkingSearcher Module Functions

• Functions contained in ParkingSearcher.js

```javascript
refreshMenu: function () {
  if ($("#" + ParkingSearcher.idMenu).length == 0) {
    $('"#pageIndexPage")
      append("<div id="" + ParkingSearcher.idMenu + "" class="commonHalfMenu"></div">
    }
  ViewManager.render(ParkingSearcher.results, "#" + ParkingSearcher.idMenu, "ParkingMenu"));
  Utility.movingPanelWithTouch("#" + ParkingSearcher.idMenu + "ExpandHandler", "#" + ParkingSearcher.idMenu);
},
```

• Checks if there is the element that will contain the html code created through the use of Mustache library.
• It is generated the html code with template ParkingMenu.mst.html and JSON ParkingSearcher.results and added to the element container.
• Finally, the feature that allows the users to widen the menu by dragging the handler is added to it
ParkingSearcher Module Functions

• Functions contained in ParkingSearcher.js

```javascript
successQuery: function (response) {
    ParkingSearcher.results = responseObject["Results"];
    ParkingSearcher.refreshMenu();
    ParkingSearcher.show();
    MapManager.addGeoJSONLayer(responseObject);
    ParkingSearcher.resetSearch();
},

errorQuery: function(error) {
    navigator.notification.alert(
        Globalization.alerts.servicesServerError.message,
        function () {} ,
        Globalization.alerts.servicesServerError.title);
},
```

These are the callbacks that should be called once the JSON, containing the data to be displayed to the user, is created. The success callback:
• will locally save the response
• will create the menu
• will show it.

If the menu will contain elements that it is possible to show on the map they will be added to the map by last function.
Before adding the logic of the new module, we create the template to be filled with the correct JSON.

This default template will simply show a menu with a header and body empty. Must have the same name as the string entered as the third parameter in the call.

ViewManager.render (ParkingSearcher.results, 
"#" + ParkingSearcher.idMenu, 
"ParkingMenu");
ParkingSearcher Module Template

• Before adding the logic of the new module, we create the template to be filled with the correct JSON.

This template will be saved in the folder called «templates». To add a title to the header we should add this item to all files labels.*. Json
ParkingSearcher Module Template

- Before adding the logic of the new module, we create the template to be filled with the correct JSON.

```html
<div id="parkingMenuHeader" class="panel panel-default" style="position: absolute; right: 0px; top: 0px; border-radius: 0px;">
  <div id="parkingMenuExpandHandler" class="grippyContainer grippyContainer-horizontal" style="text-align: center;">
    <div class="grippy grippy-horizontal"></div>
  </div>
  <div class="panel-heading" style="padding: 0px 10px; height: 52px; border: none;">
    <a class="pull-right" onclick="ParkingSearcher.hide()">
      <i class="glyphicon glyphicon-remove" style="float: right; padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
  </div>
  <div id="parkingMenuExpand" class="pull-left" onclick="ParkingSearcher.expandAndParkingSearcher();">
    <i class="glyphicon glyphicon-plus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
  </div>
  <div id="parkingMenuCollapse" class="pull-left" onclick="ParkingSearcher.collapseAndParkingSearcher();">
    <i class="glyphicon glyphicon-minus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
  </div>
  <div id="parkingMenuHeaderTitle" style="line-height: 52px; color: #333;">
    $("#parkingMenuHeaderTitle").html("Globalization.labels.parkingMenu.title")
  </div>
</div>
```
Create ParkingSearcher Module

The goal of this example is to create a **new module** that in addition to viewing the list of car parks as is already the case for the button named “Car Park" will **show directly** the **number of free parking lots** for each car park found.

In ParkingSearcher.js must be made the logic that **retrieves data** from API describer in previous presentations and creates the **JSON** to fill the **template** and generate the new menu.
ParkingSearcher Called API

- The following API returns the list of parking that are located at a maximum distance of 300 meters from the location sent. The list is limited to 100 items.

http://www.disit.org/ServiceMap/api/v1/?
selection=43.7778;11.2481&
categories=Car_park&
maxResults=100&
maxDists=0.3&
format=json&
lang=it&
geometry=true
ParkingSearcher Called API

• The returned data are not sufficient to create the final JSON, because these data are lacking on the realtime information

There are data from all car parks nearby, but there are few properties that are received
ParkingSearcher Called API

- The following API which returns all information relating to a single service

http://www.disit.org/ServiceMap/api/v1/?

serviceUri=http://www.disit.org/km4city/resource/RT04801702315PO&
format=json&
lang=it
ParkingSearcher Called API

- The returned data are not sufficient to create the final JSON, because these data are **relative to only one car park**

There are data from **one car parks nearby**, but there are **many properties** that are received.
ParkingSearcher Module Logic

• The idea is to call the first API that returns the complete list of nearby car park, and for each car park in the list call the second API that returns detailed information with the number of free parking lots.
ParkingSearcher Module Logic

• The first API can be call in the app with the following functions

```javascript
search: function(){
  var parkingQuery = QueryManager.createCategoriesQuery(['Car_park'], SearchManager.searchCenter, "user");
  APIClient.executeQuery(parkingQuery, ParkingSearcher.searchInformationForEachFeature, ParkingSearcher.errorQuery);
},
```

http://www.disit.org/ServiceMap/api/v1/?
selection=43.7778;11.2481&
categories=Car_park&
maxResults=100&
maxDists=0.3&
format=json&
lang=it&
geometry=true

The **first function** creates the string that contains the parameters from “?” to the end.

The **second function** adds the URL of the API and makes the call. When the data has been received calls the error or success callback.
ParkingSearcher Module Logic

- The second API can be called in the app with the following functions:

  For each car park listed is called the API that returns details.

  If there is no car park in the list is called a function which **doubles the radius** of the search area **until at least one car park is in the list** or the radius is greater than 200 km.
ParkingSearcher Module Logic

- The number of free parking lots is copied from a realtime object in the properties to make writing the template easier. It is also added as a property a string that identifies the text color based on the number of free parking lots.

```javascript
mergeResults: function (response) {
  for (var category in response) {
    if (response[category].features !== null) {
      if (response[category].features.length !== 0) {
        if (response.realtime !== null) {
          if (response.realtime.results !== null) {
            if (response.realtime.results.bindings[0] !== null) {
              if (response[category].features[0].properties.freeParkinglots !== null) {
                response[category].features[0].properties.freeParkinglots = response.realtime.results.bindings[0].freeParkinglots.value;
                if (response[category].features[0].properties.freeParkinglots > 20) {
                  response[category].features[0].properties.freeParkinglotsColor = "green";
                } else if (response[category].features[0].properties.freeParkinglots > 0) {
                  response[category].features[0].properties.freeParkinglotsColor = "orange";
                } else {
                  response[category].features[0].properties.freeParkinglotsColor = "red";
                }
              }
            }
          }
        }
      }
    }
  }
}
```

This function controls how many calls have already returned the details or returned error.
ParkingSearcher Module Logic

This is the function that receives the end JSON and shows it to the user, by creating the marker on the map and populating the list through the template.

The JSON is enriched with additional information such as distance from GPS or from a manual search and list is sorted according to these values.
ParkingSearcher Module Template

- This is the final template that allows you to show the user a list of car parks in its vicinity with an indication of the number of free parking lots.
ParkingSearcher in main menu

- Final version of the button with call to module logic

```json
{ "callback": "principalMenu.hide(); MapManager.centerMapOnGPS(); SearchManager.search("ParkingSearcher");" }
```

The search function of the variable SearchManager asks the user where want search (GPS, Manual or Last Service) and then call the search function of the variable which is passed as string

parkingSearcher.principalMenu.json
ParkingSearcher Module Finished
Further readings

• TC5.16. Exploiting Smart City API for developing Mobile and Web Apps
• TC5.15. Snap4City Smart City API Collection and overview, real time
• TC5.17. Search on Services via Smart City API: MicroApplication, Exploiting Micro Applications in HTML5 based on Advanced Smart City API
• TC5.18. Snap4City API are documented in Swagger, and tested in Postman
• TC5.19. Using ServiceMap as a Tools for Developing web and mobile apps and micro applications
Useful links

- **US1.** Using City Dashboards
- **US2.** Using and Creating Snap4City Applications with Dashboards
- **US3.** Using and Creating Developer Dashboards, AMMA dashboard, and/or Resource Dashboards
- **US4.** Creating City Dashboards and related Event Monitoring and Actions
- **US5.** Discovering City Services Exploiting Knowledge Base via ServiceMap
- **US6.** Developing and using processes for data transformation
- **US7.** Data Analytics and related integration aspects
- **US8.** Using the Living Lab Support tools
- **US9.** Creating Snap4City IOT Applications, different formats, protocols, brokers, communications
- **US10.** Using and Managing the Scalable Snap4City Infrastructure
- **US11.** Using tools/services of a secure and privacy respectfully solution
Former Documentation

• Documentation Smart City API
  – http://www.disit.org/6991

• Ontology and Km4City Tools:
  – Http://www.km4city.org
  – http://www.disit.org/6506 Ontology and documentation

• Snap4city is Open Source on GitHub as DISIT lab:
  – https://github.com/disit
  – https://github.com/disit/snap4city (mobile App kit)
Acknowledgements
Main running projects

- Sii-Mobility → mobility and transport, sustainability
- REPLICATE → ICT, smart City Control room, Energy, IOT
- RESOLUTE → Resilience, ICT, Big Data
- GHOST → Strategies, smart city
- TRAFAIR → Environment & transport
- MOSAIC → mobility and transport
- WEEE Life → Smart waste, environment
- Smart Garda Lake → Castelnuovo del Garda
- 5G → Industry 4.0 vs SmartCity
- Green Impact → Industry 4.0, Chemical Plant
- SmartBed (laid) → smart health
- Green Field Peas (soda) → Industry 4.0, Chemical plant
- PISA MobiMart and Agreement → data aggregation, Living Lab
- Lonato del Garda → smart parking, environment
- Herit Data → tourism, culture and management
- MobiMart → mobility and transport
- ISPRA JRC → site management and services

Snap4City (C), August 2020
Acknowledgements

Thanks to the European Commission for founding. All slides reporting logo of Snap4City [https://www.snap4city.org] of Select4Cities H2020 are representing tools and research founded by European Commission for the Select4Cities project. Select4Cities has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation Programme (grant agreement n° 688196).

TRAFAIR is a CEF project. All slides reporting logo of TRAFAIR project are representing tools and research founded by the EC on CEF programme [http://trafair.eu/]

Thanks to the European Commission for founding. All slides reporting logo of REPLICATE H2020 are representing tools and research founded by European Commission for the REPLICATE project. REPLICATE has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation Programme (grant agreement n° 691735).

Thanks to the European Commission for founding. All slides reporting logo of RESOLUTE H2020 are representing tools and research founded by European Commission for the RESOLUTE project. RESOLUTE has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation Programme (grant agreement n° 653460).

Thanks to the MIUR for co-founding and to the University of Florence and companies involved. All slides reporting logo of Sii-Mobility are representing tools and research founded by MIUR for the Sii-Mobility SCN MIUR project.

Km4City is an open technology and research line of DISIT Lab exploited by a number of projects. Some of the innovative solutions and research issues developed into projects are also compliant and contributing to the Km4City approach and thus are released as open sources and are interoperable, scalable, modular, standard compliant, etc.
DISIT thanks to

Herit Data: Tourism and Mng.  https://herit-data.interreg-med.eu/
Snap4City: IOT/IOE smart city  www.snap4city.org
Trafair: CEF project with several Cities http://trafair.eu/
Mosaic: Mobility and transport model
Km4City:  http://www.km4city.org
REPLICATE H2020, SCC1, EC flagship  http://replicate-project.eu/
Sii-Mobility SCN MIUR: http://www.sii-mobility.org
Feedback: retail and GDO Big Data analytics
5G with 3G-Wind, Open Fiber, Estra
Coll@bora Social Innovation, MIUR:  http://www.disit.org/5479
RESOLUTE H2020, EC:  http://www.resolute-eu.org
TRACE-IT, RAISSS, TESYSRAIL, ...
Mobile Emergency:  http://www.disit.org/5404
Further readings

- HOW TO: create a Dashboard in Snap4City
- HOW TO: add a device to the Snap4City Platform
- HOW TO: add data sources to the Snap4City Platform
- HOW TO: define privacy rules for personal data, produced by the end-users own device
- HOW TO: Develop Smart Applications, Snap4City development Life Cycle
- HOW TO: HLT vs Ingestion, and HLT vs Widgets
- HOW TO: Develop an IOT Application for Data Ingestion
- HOW TO: Upload data into Knowledge Base, ServiceMap (triple upload)
- HOW TO: Create as set of Devices with BulkProcessing
- HOW TO: Create an IOT Device Model
- HOW TO: Create an IOT Device Instance from IOT Directory tool
Snapt4City (C), August 2020

Be smart in a SNAP!

100% OPEN SOURCE

CONTACT

DISIT Lab, DINFO: Department of Information Engineering
Università degli Studi di Firenze - School of Engineering
Via S. Marta, 3 - 50139 Firenze, ITALY
https://www.disit.org

www.snap4city.org

Email: snap4city@disit.org
Office: +39-055-2758-515 / 517
Cell: +39-335-566-86-74
Fax: +39-055-2758570