



Dev App Kit for mobile and web App





- **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>

Mustache JS

Template →

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

JSON →

```
var data = {
  risultato: true,
  titolo: Città italiane,
  descrizione: Lista delle città italiane,
  citta: [
    {nome: Milano, sigla: MI},
    {nome: Roma, sigla: RM}
  ]
};
```

Mustache JS

Template →

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

JSON →

```
var data = {
  risultato: true,
  titolo: Città italiane,
  descrizione: Lista delle città italiane,
  citta: [
    {nome: Milano, sigla: MI},
    {nome: Roma, sigla: RM}
  ]
};
```

Template + JSON + Mustache

Città italiane

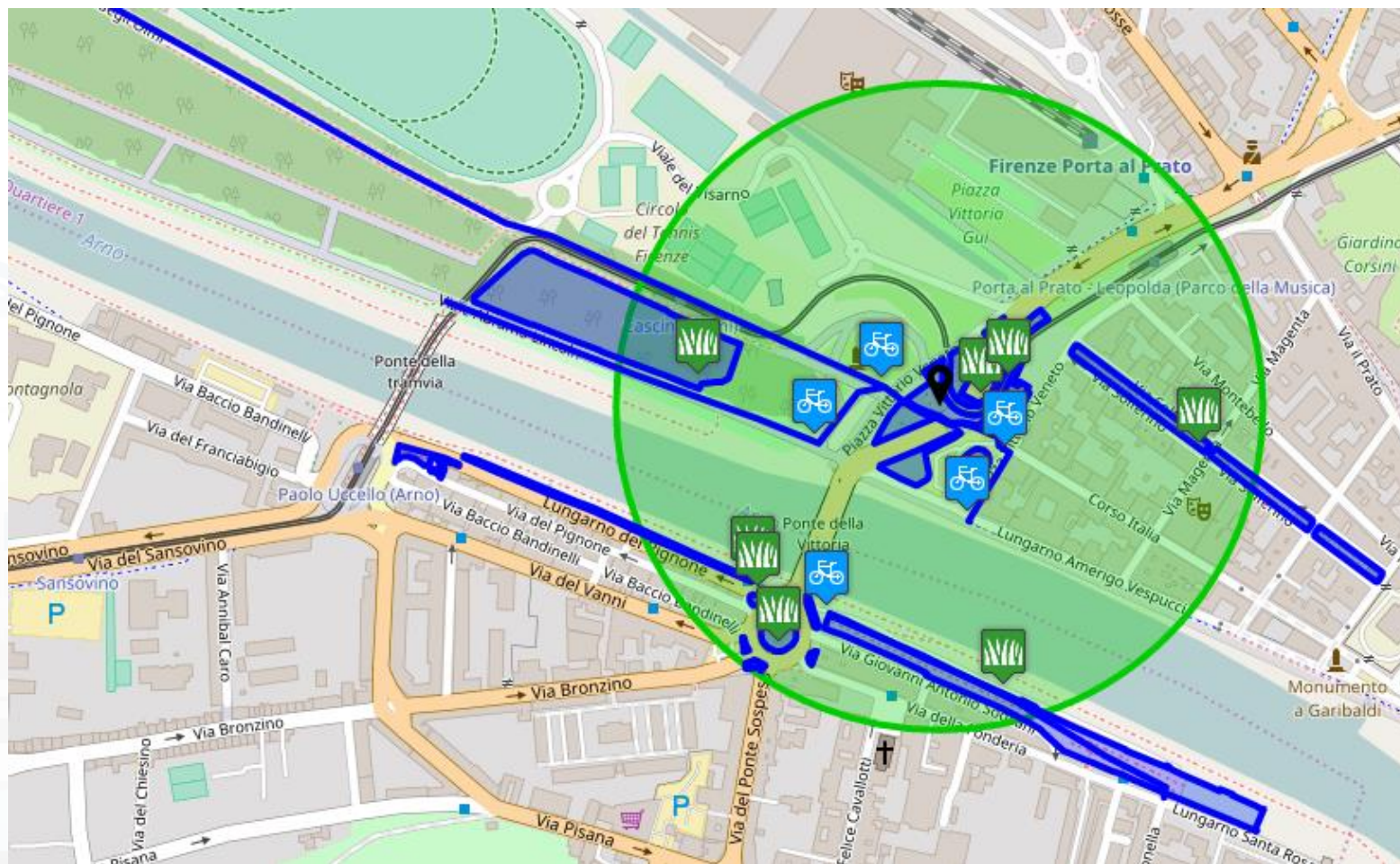
Lista delle città italiane

- Milano (MI)
- Roma (RM)

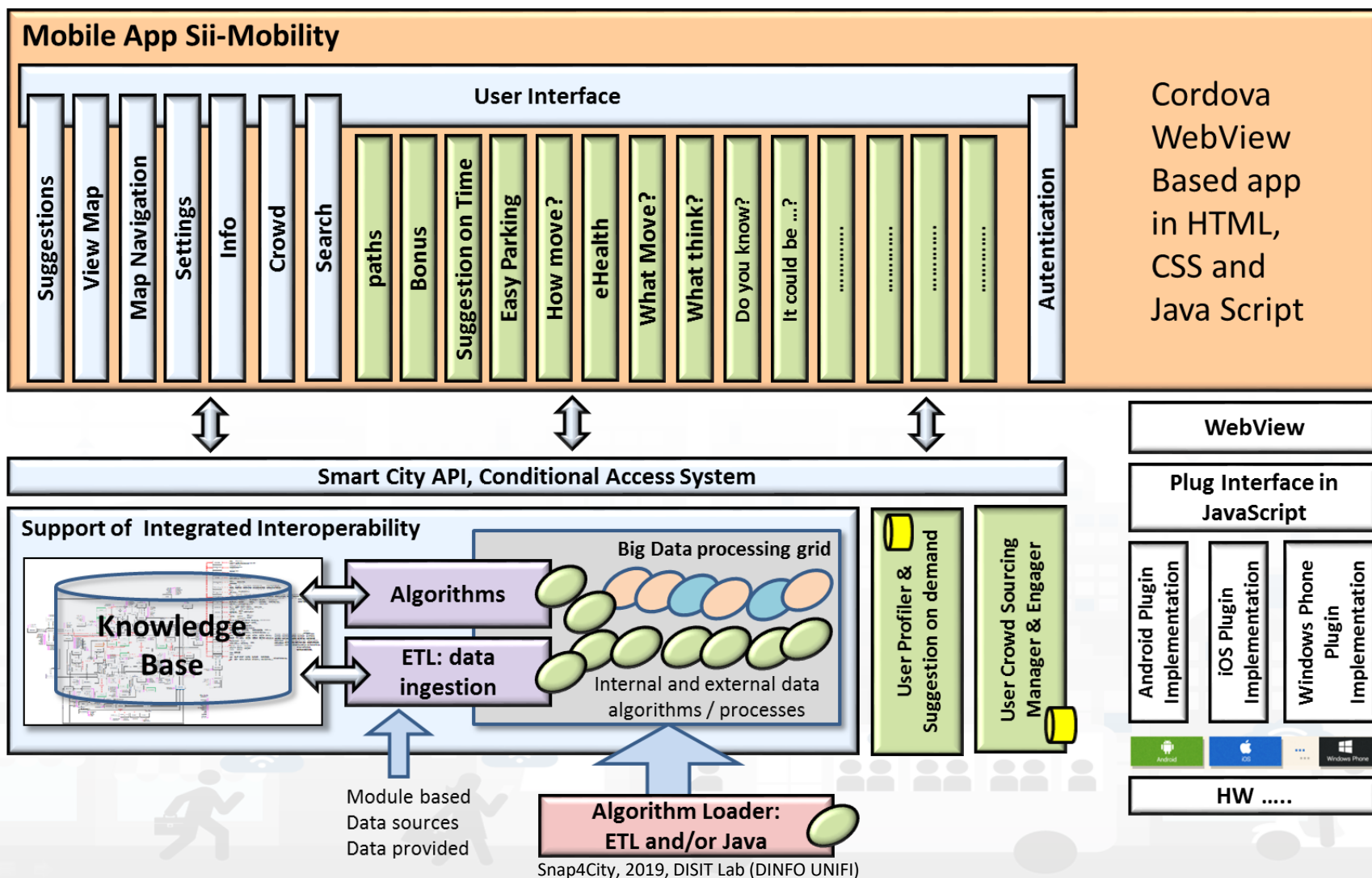


- **OpenLayers** is an open source **JavaScript library** for **displaying map data** in web browsers and can be used with a hybrid application developed with Cordova
- In the **early versions** of the app, the map was managed by **Leaflet.js** library. This was replaced because it didn't support the 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>**

OpenLayers 3.0



General architecture




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

 ExampleModule.js
Tipo: File JavaScript

 exampleModule.labels. *.json
Tipo: JSON File

 exampleModule.principalMenu.json
Tipo: JSON File

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**

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



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**

```
[
{
  "callback": "PrincipalMenu.hide(): MapManager.centerMapOnGps():".
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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
}
]
```

parkingSearcher.principalMenu.json

This field contains the **callback** for the new module.

The present callbacks should be left, because they serves 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**

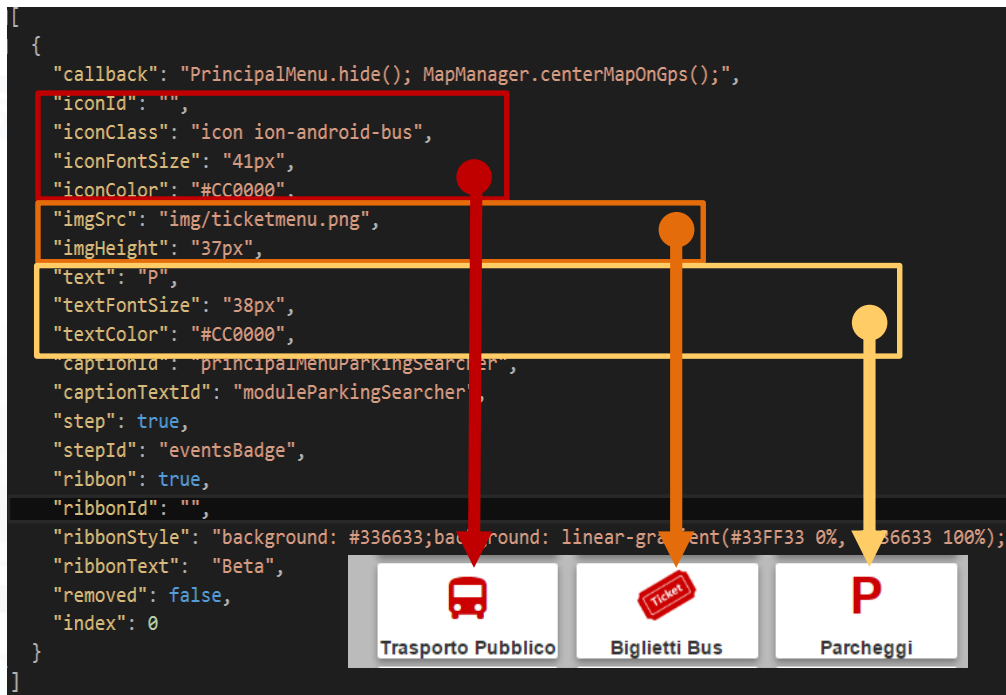
```
[
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "textColor": "#CC0000",
  "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
}
]
```

parkingSearcher.principalMenu.json

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**



parkingSearcher.principalMenu.json

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**

```
[
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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
}
]
```

parkingSearcher.principalMenu.json

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 in main menu

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

```
[
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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
}
]
```

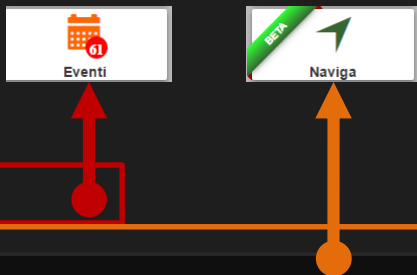
These blocks of fields are used to show the user **badges containing information** related to the button on which are located

parkingSearcher.principalMenu.json

ParkingSearcher in main menu

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

```
[
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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
}
]
```



These blocks of fields are used to show the user **badges containing information** related to the button on which are located

parkingSearcher.principalMenu.json

ParkingSearcher in main menu

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

```
[
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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.

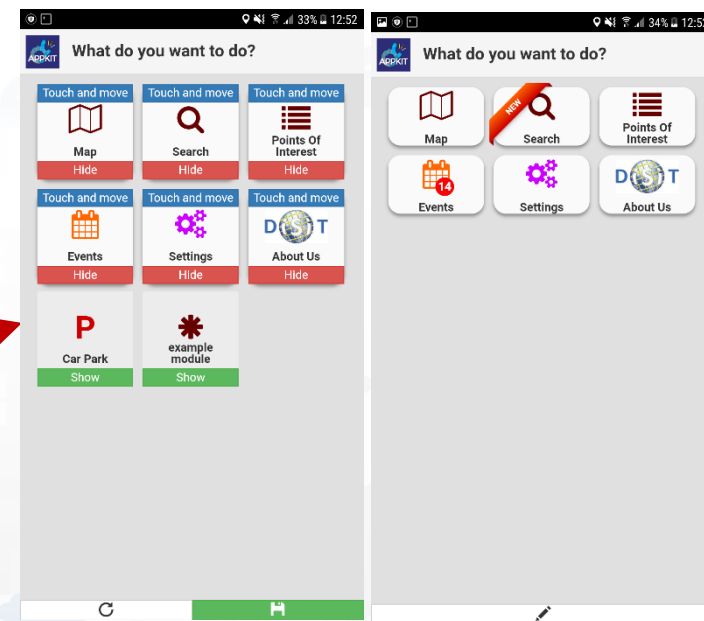
parkingSearcher.principalMenu.json

ParkingSearcher in main menu

- Field descriptions for creating buttons in the main menu

```
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps();",
  "iconId": "",
  "iconClass": "icon ion-android-bus",
  "iconFontSize": "41px",
  "iconColor": "#CC0000",
  "imgSrc": "img/ticketmenu.png",
  "imgHeight": "37px",
  "text": "P",
  "textFontSize": "38px",
  "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
}
```

parkingSearcher.principalMenu.json



ParkingSearcher in main menu

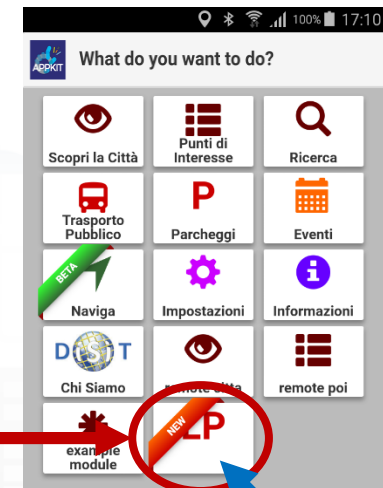
- Loading **new buttons modules** within the main menu, takes place by **comparing the captionId** field.
- If the menu already has a button with the **same captionId**, the first is **replaced** with the **new one**.
- To **remove** a button from the main menu (field **removed** hides it) add a **delete** field with value equal to **true**.

ParkingSearcher in main menu

- First version of the button

```
[
{
  "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

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

```
{
  "principalMenu": {
    "moduleParkingSearcher": "Lista de Aparcamiento"
  }
}
```

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

```
[
{
  "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(#CC0000 49%, #CC0000 49% 51%, #CC0000 51% 53%, #CC0000 53% 55%, #CC0000 55% 57%, #CC0000 57% 59%, #CC0000 59% 61%, #CC0000 61% 63%, #CC0000 63% 65%, #CC0000 65% 67%, #CC0000 67% 69%, #CC0000 69% 71%, #CC0000 71% 73%, #CC0000 73% 75%, #CC0000 75% 77%, #CC0000 77% 79%, #CC0000 79% 81%, #CC0000 81% 83%, #CC0000 83% 85%, #CC0000 85% 87%, #CC0000 87% 89%, #CC0000 89% 91%, #CC0000 91% 93%, #CC0000 93% 95%, #CC0000 95% 97%, #CC0000 97% 99%, #CC0000 99% 100%);",
  "ribbonText": "NEW",
  "removed": false,
  "index": 0
}
]
```

parkingSearcher.principalMenu.json

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

```
{
  "principalMenu": {
    "moduleParkingSearcher": "Lista de Aparcamiento"
  }
}
```

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


Labels of ParkingSearcher

- Description of **label.*.json** files



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

Tipo: File JavaScript

TODO



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 Module Functions

- Functions contained in **ParkingSearcher.js**

```
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**

```
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**

```
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**

```
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);  
},
```

- 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**

```
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);
},
```

- 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**

```

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

ParkingSearcher Module Template

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

```
<div id="parkingMenuHeader" class="panel panel-default" style="position: absolute;right: 0px;left: 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-left: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuExpand" class="pull-left" onclick="ParkingSearcher.expandParkingSearcher();">
      <i class="glyphicon glyphicon-plus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuCollapse" class="pull-left" onclick="ParkingSearcher.collapseParkingSearcher();">
      <i class="glyphicon glyphicon-minus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <b id="parkingMenuHeaderTitle" style="line-height: 52px;color: #333;">
      <script>
        $("#parkingMenuHeaderTitle").html(
          Globalization.labels.parkingMenu.title)
      </script>
    </b>
  </div>
</div>
<div id="parkingMenuInner" class="commonHalfMenuInner">
</div>
```

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");

ParkingMenu.mst.html

ParkingSearcher Module Template

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

```


<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-left: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuExpand" class="pull-left" onclick="ParkingSearcher.expandParkingSearcher();">
      <i class="glyphicon glyphicon-plus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuCollapse" class="pull-left" onclick="ParkingSearcher.collapseParkingSearcher();">
      <i class="glyphicon glyphicon-minus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <b id="parkingMenuHeaderTitle" style="line-height: 52px;color: #333;">
      <script>
        $("#{#parkingMenuHeaderTitle}").html(
          Globalization.label.parkingMenu.title)
      </script>
    </b>
  </div>
</div>
<div id="parkingMenuInner" class="commonHalfMenuInner">
</div>


```

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

```

{
  "principalMenu": {
    "moduleParkingSearcher": "Lista Parcheggi"
  },
  "parkingMenu": {
    "title": "Parcheggi"
  }
}

```

templates/ParkingMenu.mst.html

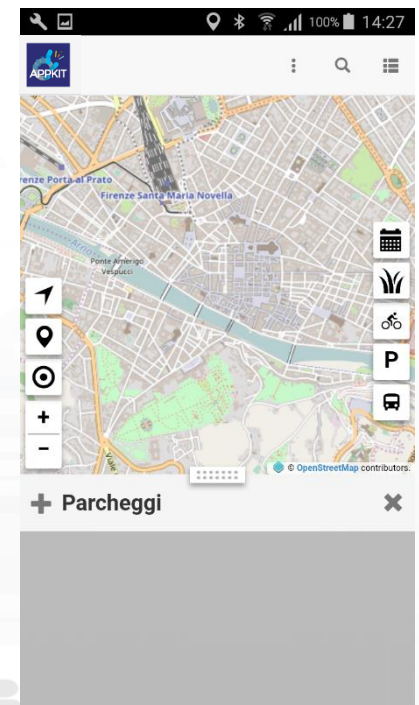
ParkingSearcher Module Template

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

```
<div id="parkingMenuHeader" class="panel panel-default" style="position: absolute;right: 0px;left: 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-left: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuExpand" class="pull-left" onclick="ParkingSearcher.expandParkingSearcher();">
      <i class="glyphicon glyphicon-plus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <a id="parkingMenuCollapse" class="pull-left" onclick="ParkingSearcher.collapseParkingSearcher();">
      <i class="glyphicon glyphicon-minus" style="padding-right: 8px; color: #777; line-height: 52px;"></i>
    </a>
    <b id="parkingMenuHeaderTitle" style="line-height: 52px;color: #333;">
      <script>
        $("#parkingMenuHeaderTitle").html(
          Globalization.labels.parkingMenu.title)
      </script>
    </b>
  </div>
</div>

<div id="parkingMenuInner" class="commonHalfMenuInner">
</div>
```

templates/ParkingMenu.mst.html



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

```

▼ object {1}
  ▼ Services {3}
    fullCount : 5
    type : FeatureCollection
    ▼ features [5]
      ▼ 0 {4}
        ▼ geometry {2}
          type : Point
          ► coordinates [2]
          type : Feature
          ▼ properties {7}
            name : Garage La Stazione Spa
            tipo : Parcheggio_auto
            typeLabel : Parcheggio auto
            serviceType : TransferServiceAndRenting_Car_park
            hasGeometry : ☐ false
            serviceUri : http://www.disit.org/km4city/resource/RT04801702315PO
            multimedia : {value}
          id : 1
        ► 1 {4}

```

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**

```

▼ object {2}
  ▼ Service {2}
    type : FeatureCollection
    ▼ features [1]
      ▼ 0 {4}
        ► geometry {2}
          type : Feature
          ► properties {26}
            id : 1
        ► realtime {2}
          ► head {2}
          ▼ results {1}
            ▼ bindings [1]
              ▼ 0 {6}
                ► capacity {1}
                ▼ freeParkingLots {1}
                  value : 282
                ► occupiedParkingLots {1}
                ► occupancy {1}
                ► status {1}
                ► updating {1}

```

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

```
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 call in the app with the following functions

```
searchInformationForEachFeature(response) {
  for (var category in response) {
    if (response[category].features.length != 0) {
      ParkingSearcher.responseLength = response[category].features.length;
      ParkingSearcher.temporaryResponse = {
        "Results": {
          "features": [],
          "fullCount": ParkingSearcher.responseLength,
          "type": "FeatureCollection",
        }
      };
    }
  };
  Loading.showAutoSearchLoading();
  for (var i = 0; i < response[category].features.length; i++) {
    var serviceQuery = QueryManager.createServiceQuery(response[category].features[i].properties.serviceUri, "app");
    APIClient.executeQueryWithoutAlert(serviceQuery,
      ParkingSearcher.mergeResults,
      ParkingSearcher.decrementAndCheckRetrieved);
  }
  SearchManager.startAutoSearch(ParkingSearcher.varName);
},
}
```

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 realtime object in the properties** to make writing the template easier. Is also added as a property a string that identifies the **text color** based on the number of free parking lots

```
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.realtime.results.bindings[0].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";
                                }
                            }
                        }
                    }
                }
            }
        }
        ParkingSearcher.temporaryResponse["Results"].features.push(response[category].features[0]);
    }
}

ParkingSearcher.decrementAndCheckRetrieved();
},

decrementAndCheckRetrieved: function(){
    ParkingSearcher.responseLength--;

    if (ParkingSearcher.responseLength == 0) {
        ParkingSearcher.successQuery(ParkingSearcher.temporaryResponse);
        Loading.hideAutoSearchLoading();
    }
},
```

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

ParkingSearcher Module Logic

```

successQuery: function (response) {
    var responseObject = response;

    if (SearchManager.typeOfSearchCenter == "selectedServiceMarker") {
        MapManager.searchOnSelectedServiceMarker = true;
    }
    for (var i = 0; i < responseObject["Results"].features.length; i++) {
        responseObject["Results"].features[i].id = i;
        Utility.enrichService(responseObject["Results"].features[i], i);
    }
    if (responseObject["Results"].features[0].properties.distanceFromSearchCenter != null) {
        responseObject["Results"].features.sort(function (a, b) {
            return a.properties.distanceFromSearchCenter - b.properties.distanceFromSearchCenter
        });
    } else {
        responseObject["Results"].features.sort(function (a, b) {
            return a.properties.distanceFromGPS - b.properties.distanceFromGPS
        });
    }

    ParkingSearcher.results = responseObject["Results"];
    ParkingSearcher.refreshMenu();
    ParkingSearcher.show();
    MapManager.addGeoJSONLayer(responseObject);
    ParkingSearcher.resetSearch();
},

```

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**

```
<!-- {{#feature}} {{#properties}}-->
<div class="panel panel-default" style="margin: 0px 5px 10px 5px; color: #000; text-decoration: none;" onclick="InfoManager.showInfoAboutOneMarker('{{#properties}}')">
  <div class="panel-body card-2" style="position: relative">
    {{#distanceFrom}}
    {{#freeParkingLots}}<b style="position: absolute; bottom: 0px; right: 10px; font-size: 24px; color: {{#freeParkingLotsColor}}">
      {{#freeParkingLots}}</b></div>
    {{#imageFrom}}{{#imageThumb}}
    
    <b style="text-align: center;">{{#unescapeHtml}}{{#name}}{{#unescapeHtml}}</b>
    {{#typeLabel}}<br><b id="parkingMenuTypeLabel{{#identifier}}">
      <script>$("#parkingMenuTypeLabel{{#identifier}}").html(Globalization.labels.infoMenu.type)</script>
    </b>{{#typeLabel}}</div>
    {{#distanceFromSearchCenter}}<br><b id="parkingMenuTextSearchDistanceFromSearchCenter{{#identifier}}">
      <script>$("#parkingMenuTextSearchDistanceFromSearchCenter{{#identifier}}").html(Globalization.labels.textSearchMenu.distanceFromSearchCenter)</script>
    </b>{{#distanceFromSearchCenter}} m</div>
    {{#distanceFromGPS}}<br><b id="parkingMenuTextSearchDistanceFromGPS{{#identifier}}">
      <script>$("#parkingMenuTextSearchDistanceFromGPS{{#identifier}}").html(Globalization.labels.textSearchMenu.distanceFromGPS)</script>
    </b>{{#distanceFromGPS}} m</div>
  </div>
</div>
<!-- {{/feature}} -->
```

ParkingSearcher in main menu

- Final version of the button with call to module logic

```
{
  "callback": "PrincipalMenu.hide(); MapManager.centerMapOnGps() SearchManager.search('ParkingSearcher');",
  "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
}
```

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



- [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](#)

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)