UNIVERSITE CATHOLIQUE DE LOUVAIN

ECOLE POLYTECHNIQUE DE LOUVAIN

UCLCampus: a mobile application for UCL students

Supervisor: Yves Deville

Readers: Kim Mens
Hildeberto Mendonça
Mathieu Zen
Jorge Perez Medina

Thesis submitted for the Master's degree
in computer science and engineering
by : Baptiste Lacasse
Arnold Moyaux

Louvain-la-Neuve

Academic year 2015-2016

Contents

Chapter 1

Introduction

Brief introduction of the project, the goals and the contents of the rest of the thesis.

Chapter 2

Background

In this section, we will look at the different existing technologies relating to the different aspects of our project and will explain the choices we made.

2.1 Cross-platform mobile development tools

2.1.1 The native approach

2.1.2 Our choice

2.2 Open-source project and code sharing

2.3 Project Management Methodologies

Here we detail the choices we made as to how we were going to manage the different parts of the project.

Chapter 3

Functionalities of UCLCampus

In this chapter, we will show how we defined the relevant functionalities of our application as well as the user interface.

3.1 Choice of functionalities and sections

In order to define what kind of functionalities we wanted to be part of our application, we needed to know what the students needed. The first step was to define a number of user stories that we would then translate into functionalities.
We split our user stories into several categories:

- Studies: anything directly related to a user's studies, for instance his classes, the lecture halls or the libraries.
- Campus: anything related to student life in the campus but not related to the user's studies. For instance "Kot à Projets" or "Cercles".
- City: anything related to the city the user is in but not related to the university. For instance a cinema or restaurants.
- Tools: the tools offered by the application that might relate to several other categories. For example the map.
- Settings: the settings of the application. For example the language or the currently selected campus.

We also define two types of users:

- Students: students can access all the functionalities of the application using their UCL login information. Indeed, some functionalities are student specific. For instance, it wouldn't make sense for a person who isn't a student to try to access his or her schedule.

- Users: users are people who aren't students but might still be interested in some functionalities the application has to offer.

In our user stories, any story starting by 'as a student' cannot be used by users while any story starting by 'as a user' can be used by both users and students.

We will now give a list of the different user stories we thought of for each of our categories.

Studies

- Schedule
  - As a student, I can access my schedule in order to know when my courses are given.
  - As a student, I want to know where a course is given.
  - As a student, I want to know the name of a teacher giving a certain course of my schedule.
  - As a student, I can export my schedule to my phone's agenda so that I don't need Internet access to see it.
- Libraries
  - As a user, I can see whether a library is open or closed.
  - As a user, I can display the address of any library.
  - As a user, I can have a GPS guide to access libraries from my location.
- Lecture halls
  - As a user, I can check the address of any lecture hall.
  - As a user, I can have a GPS guide to access lecture halls from my location.
- Websites
  - As a user, I can quickly access the moodle website through the application.
  - As a user, I can quickly access the UCL website through the application.

Campus

- Events
  - As a user, I can see a list of events taking place in my campus.
  - As a user, I can sort the events by category.
- Kots à Projet
  - As a user, I can check Kots à Projet to know their address and projects.

  - As a user, I can have a GPS guide to access Kots à Projet from my location.

- Cercles
  - As a user, I can check "Cercles" to know their address.
  - As a user, I can have a GPS guide to access "Cercles" from my location.

- Restaurants Universitaires
  - As a user, I can see the different "Restaurants Universitaires" in my campus.
  - As a user, I can check the menu of the "Restaurants Universitaires".
  - As a user, I can have a GPS guide to access "Restaurant Universitaires" from my location.

- Sports
  - As a user, I can see a list of sports organized in my campus.
  - As a user, I can sort the sports by day or by sport.

City

- Tourism
  - As a user, I can see the address of the city's information center.
  - As a user, I can see a list of the museums of the city I'm in.
  - As a user, I can see whether a museum is opened or closed.

- Activities
  - As a user, I can see the address of the city's cinema in order to access it with the help of a GPS guide.
  - As a user, I can see several activities I can do in the city I'm in.

- Restaurants and bars
  - As a user, I can see a list of the restaurants of the city I'm in.
  - As a user, I can see a list of the bars of the city I'm in.

Tools

- Maps
  - As a user, I can access a map of the city I'm in in order to check points of interests.
  - As a user, I can receive help from a GPS guide in order to access a location of my choice on the map.

- Mail
  - As a student, I can check my emails on my uclouvain account.
  - As a student, I can send emails from my uclouvain account.
- Help
  - As a user, I can see where the UCL parkings are located.
  - As a user, I can receive help about how to configure the UCL wifi.
  - As a user, I can receive help about common transportation.

Settings

- As a user, I can change the application's language to French, English or Dutch.
- As a user, I can select my campus.
- As a student, I can login on my UCLouvain.be account.
- As a student, I can log out from my UCLouvain.be account.

3.2 User interface

Once we determined the different features we wanted in our application, we needed to organize them in a way that makes sense for users. In order to do so, we made sketches of what the application might look like using InVision. InVision is a website that lets people design and style mobile applications prototypes. It allows us to get an idea of what the finished product might look like without having to dive into any code. The sketches we made are available in the annex.

In these sketches, we can see that we decided to have one menu per category we defined in the previous section. Each menu has an associated color, allowing the user to always have visual clues to help them know where they are.

Once the sketches were done, we shared the link to our prototype to over 1000 students, mostly student in their first year, as they represent the future users of this application. They were able to browse through the application using the buttons, as if it was already working, and leave comments and feedback wherever they wanted to.

While we didn't receive as much feedback as we would have hoped, the one we received was constructive and helpful. Most people were satisfied with our 3 first categories, Studies, Campus and City. The fourth category, however, was more criticized. Here are some comments we received concerning the Tools category.

After reading these comments, we decided to rework the Tools section. We agreed that the mail part was superfluous and we decided to drop it entirely. We also decided to drop the "Help" section as we didn't think it was important enough. That left us with the map. We decided that it was important that the map was not grouped with the city, the studies or the campus as it was important to all three sections. We thus decided to leave it in the Tools section. We also decided against renaming the section "Maps" as future contributors may very well add functionalities we didn't think of in this section.

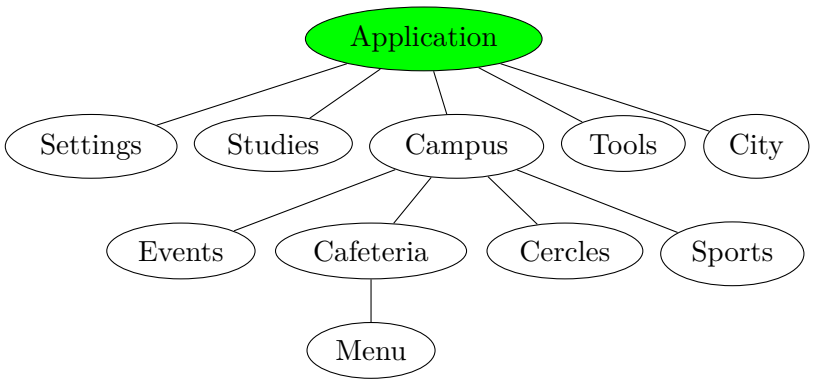
Chapter 4

Implementation

Here we will explain the overall architecture of the application. We will also explain some aspects we considered when implementing the application.

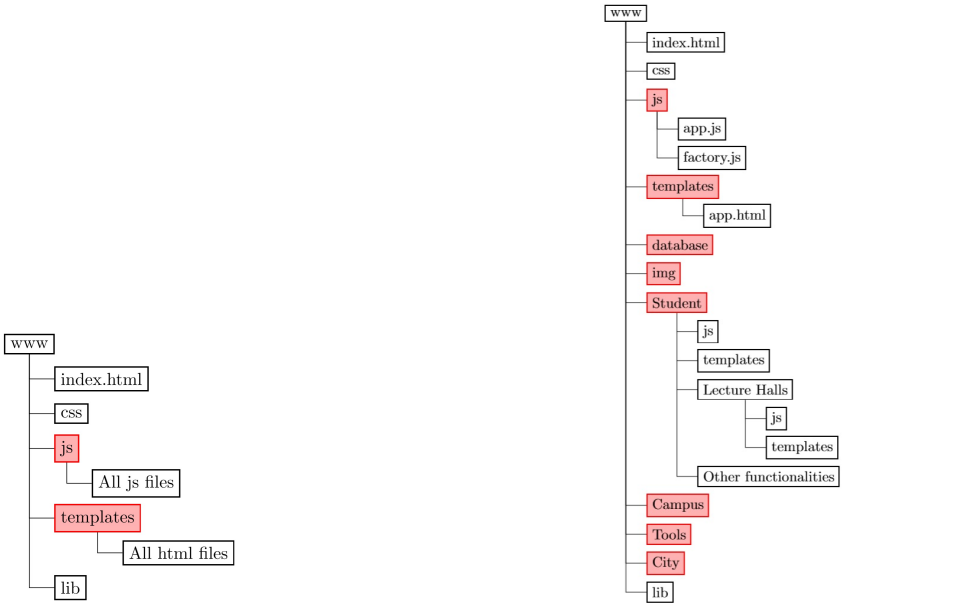
4.1 Architecture

The purpose of the application is to be extensible and easily maintainable. We wanted a programmer to have the possibility to add his functionalities at each level of the application. For this we thought our implementation as a tree. At the top level we have the generic parts and configurations that will be everywhere in the application under it we have the settings menu and next to it we have four branches pointing to global sections that we decided to create, themselves pointing to their functionalities and so on. Here is a part of the tree in order to give you the idea.



4.1.1 Folder organisation

We wanted to keep the same state of mind for the file organisation. Ionic base architecture is to put all html file in a folder named templates, all js in a JS folder, ... The problem is that become messy once we have a lot of functionalities(thus a lot of files). We modify it to respect the tree architecture we want. With our folder system, a programmer can add his own subtree to the main tree. And if you want to modify a specific functionality, you have directly access to the related files. Figure 4 1 is a summary of the change, red folder are those we modify.



(a) Before (b) After

Figure 4.1: Folder evolution

4.1.2 Information processing

Here I just explain how we deal with the information processing from an architecture's design point of view. The section 4.4 explains in detail how we did it for each specific part. There is a lot of external information to relay into the application (libraries schedule, libraries addresses, daily events in the campus,...). We have two possible ways to import them.

Database

Pros	Cons
Always available	Need someone to update
Easy information retrieval(query)	Takes memory
Easy to modify	
Fast	

Table 4.1: Pros and cons of a database

Web parsing

Pros	Cons
Automatic update	Need an Internet connexion and an operational server side
Easy information retrieval with web services(query)	Horrible information retrieval without web services
No hardware memory consumption	If the web server change, maybe you will need to recode all the parsing method
	Slow

Table 4.2: Pros and cons of the web parsing

An considerable limitation is the need for someone in order to update the database or creating new parsing system. We have no workforce for it, so if we have the choice between the two methods, we will select the one needing less modification in the future.

4.1.3 Front-end and back-end

- Front-end: Part of the user interface that can be separated in two fields. First is design and the second is html, css and JavaScript development.
- Back-end: Is the hidden part of the iceberg, what you can't see. For example: the database, the parsing function, ...

We create a front-end and a back-end system in our application. It helps a lot for the maintain because you can modify part without involving the other. For example, we store lecture halls in the database but the UCL create a new website with web services providing all lecture halls and their information. It's better than the database because it automatically updates and so you want to use them instead. You can do it in a specific part that is totally isolated from the code for the user interface.

4.1.4 Factory

Factory is a functionality from angularjs which we used as a back-end service. The factory will take the data from the database or web parsing, modify the data format to be easier to handle and then transmit it to the front-end. Figure 4.2 illustrates this.

For example, we open the page for the schedule. (1)the system notifies the schedule factory in order to get his data. (2) the factory create a custom HTTP request and send it to the ADE

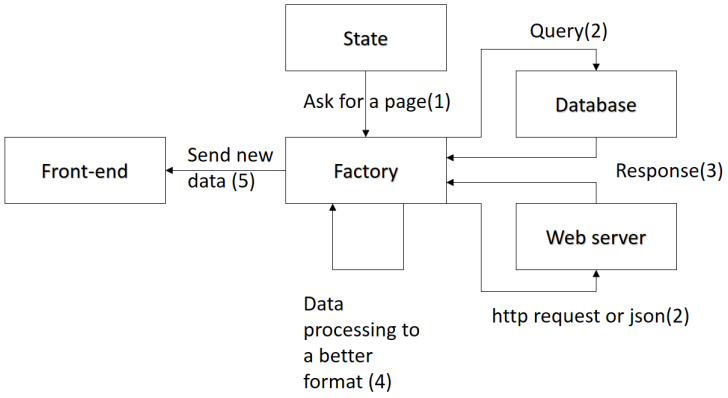


Figure 4.2: Factory operation

domain. (3) Server sends a response. (4) Data in the response are not easy to manipulate (long string with html tags inside), so the factory pick the important element in the response(with a parsing algorithm) and put them in a JavaScript object where data are easy to handle. (5) Send new data to front-end.

4.1.5 State provider

The state provider is a functionality of angularjs. It allow you to define the page configuration (cached or not for example). Furthermore it create an edge between the back end and the front end, figure 4.3 illustrate the operation of the state provider.

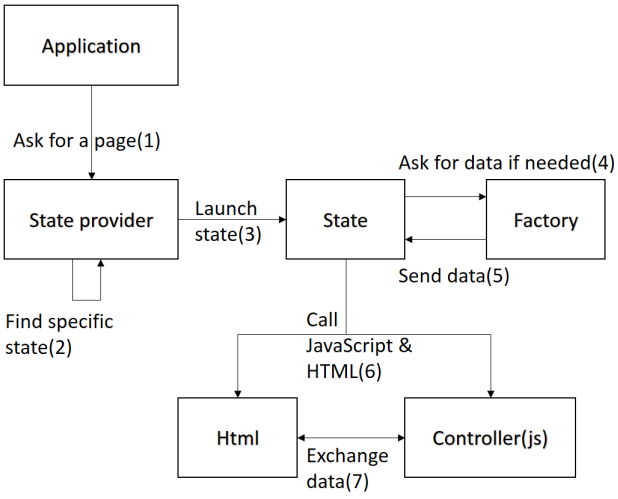


Figure 4.3: State provider operation

First the application ask for a page. The state provider contain a list of specific state(one state by page), it will return the state relative to the asked page. The state will contain information about the page:

- The location of the html template
- The associate controller(js)
- A flag for cache memory(true or false)
- The section membership(for example student if we are in the student tab)

. For back end data(optional), the state will call the needed factory and wait its response before launching the template and the controller for the user interface. JavaScript(controller) and Html can exchange informations and sent it to back end in some cases.

4.2 Coding standards

4.2.1 Header and footer

We managed our code to create automatically a default new footer and header when you start new html template. The footer and header have a specific color for each tab-section (blue for student, yellow for campus, purple for tools and green for town). We allow developers to overwrite the default header but under two condition

- The title have to keep the same font and have to be center.
- The header color must respect the section color

Footer must not be overwrite or be hidden. If developers need to insert their specific header or footer, Ionic provide a css in order to add subfooter and subheader.

4.2.2 Tree hierarchy

We want developers to keep the tree hierarchy we created for the state and for the folder. For css and extra JavaScript, html, we don't enforce any rules as long as it stay in the relative module.

4.2.3 Colors

We used the UCL colors code[1]. For this we modified the sass theme of Ionic. This allow module developed with basic ionic css component to update with the application color instantly and automatically. The main color of UCLCampus is the blue "UCL".

Color	css name	code
Blue	positive	#00214e
Yellow	energized	#f29400
Purple	royal	#88005d
Green	balanced	#76ad1c

[1]https://www.uclouvain.be/459461.html

4.3 Security

TODO

4.4 Information retrieval

In this section we will detail the different sources of information we used in our application as well as the technologies we had to use in order to retrieve said information.

4.4.1 Student data(OSIS)

TODO

4.4.2 Libraries and lecture halls

In order to retrieve a list of all libraries and lecture halls in the university, we contacted the UCL to ask them if such a database existed. Sadly, it appeared that no database of this kind exists. We then wondered how LLNCampus managed to obtain all the information they had on lecture halls and libraries. Since their application is also open-source, we were able to find the database they used. Since it was very complete and was the only database available, we decided to re-use this database. However, since their database was built with only one campus in mind, we had to change it a bit.

Database structure

The database is an SQLite database. The table we use for the most part is called "poi" (point of interest) and has the following fields:

- ID (integer): used to identify the item
- Name (text): name of the point of interest
- Latitude (real): latitude of the point of interest
- Longitude (real): longitude of the point of interest
- Type (text): type of the point of interest, can be "auditoire", "kap", "bibliotheque", ...
- Address (text): address of the point of interest
- Img (text): path to the image linked to the poi

To these fields we added the following two:

- Campus (text): campus where the poi is located

- Abbr (text): abbreviated version of the poi. For instance "BARB" for the Ste-Barbe lecture hall

The first field is useful if we want to have different campuses while the second is used to find the poi on the map.

Retrieving lecture halls

Once we had the completed the database, we had to connect it to our application. In order to do so, we used the cordovaSQLite plugin. Once connected, we were able to retrieve the lecture halls using the following SQL query:

```
SELECT * FROM poi WHERE TYPE = 'auditoire' AND CAMPUS = ?
```

This query will retrieve all the fields from the poi table we discussed earlier which have the type "auditoire" and are located in the desired campus. The "?" character is used as a placeholder for an argument we can pass to the function. We will then store the result of the query in a JavaScript array and used that array to display the information to the user.

Retrieving libraries

In order to retrieve all the information concerning libraries, the poi table was not enough. Indeed, we wanted the users to be able to know whether a library was opened or closed and the schedule was not available in the poi table. Thankfully this information was available in another table, called "bibliotheque_horaire". This table only has 4 fields:

- Building_ID (integer): identifies the library. Same id as the one used in the poi table.
- Day (integer): Day of the week (0 for monday, 1 for tuesday, ...)
- Begin_time (integer): Time of the day when the library opens. Given in minutes elapsed since midnight
- End_time (integer): Time of the day when the library closes. Given in minutes elapsed since midnight

Given this new table we were able to retrieve all the information we needed using the following SQL query:

```
SELECT * FROM poi, bibliotheque_horaire WHERE poi.TYPE = 'bibliotheque'
AND poi.ID == bibliotheque_horaire.BUILDING_ID AND DAY = ? AND CAMPUS = ?
```

This query retrieves all poi with type "bibliotheque" and finds their respective schedule for a given day, all that for a given campus. The result in then stored in a JavaScript array.

4.4.3 Events

In order to find events taking place in Louvain-la-Neuve, we used the www.louvainfo.be website. This website provides a schedule of all upcoming events in the city. It is however specific to Louvain-la-Neuve and we would need to find equivalent websites for the other campuses.
The website provides an Atom RSS feed of the events. We decided to use Yahoo Query Language to parse the RSS feed and retrieve the information. The reason we used the Yahoo Query language was that it was the only way we found to perform such an operation. Indeed, the more widely used Google Feed API has been deprecated. We therefore had no other choice but thankfully we haven't encountered any issue with YQL.

We used the following query to retrieve the feed:

```
select * from feednormalizer where url='http://louvainfo.be/evenements/feed/calendar/'
```

The result is then parsed and stored in a JavaScript array.

4.4.4 Schedule

For the schedule we had no choice between database or web parsing. The ADE dump database for all courses(of all the students) has a size of 70mb which is way too much memory for a mobile application. Therefore ADE has a custom web site where you are able to send customize url in order to extract specific informations. It's a bit worse than web services because you need a parsing technique.

ADE custom URL

The request always start with `http://horairev6.uclouvain.be/jsp/custom/modules/plannings/direct_planning.jsp?`. After it we can add some query parameters.

- code : The course code we want to see on the schedule. We picked the logged student courses codes for it.
- weeks : The weeks we want to see. It's a number per weeks. We wanted to get all the year so we used a suite from 1 to 52 (1,2,3,...).
- projectID : A number defining the academic year we want to see(16 for 2015-2016). This number need to be manually updated each year. We couldn't update it automatically because it seems to be a random number chosen each year(7 and 16 for the last two years).
- password and username : Require for ADE connexion.
- page configuration : There are multiple parameters that I won't explain. Their purpose is to custom the ADE web page we access. We used it to get a tabular summarizing the schedule that is easier to parse than the original web site.

Parsing

Http response contain an html code. The tabular is really useful here in the parsing because it provides a block of informations easy to extract. It contain information about each lecture of the year. Here is the syntax of the tabular in the response.

```
<tbody>
  <tr>
    <td>Date</td>
    <td>Course Code</td>
    <td>Hours:Minutes</td>
    <td>Date</td>
    <td></td>
    <td>Useless information</td>
    <td>Eleves</td>
    <td>Professors</td>
    <td>Place</td>
    <td>Course Code again</td>
  </tr>
  <tr> an other lecture </tr>
  ...
</tbody>
```

We used a string parsing based on regular expression in order to extract information from tags and place it into a JavaScript object with field easily accessible. At the end the back end sent a list of lecture object to the front end.

Local storage

We thought it could be good to keep the schedule relative to the logged student in memory. In this case he could have an access to it every time without the need of an internet connexion. For this, once the parsing done, we keep the new schedule object in memory and the next time the student access the page we load it from local storage instead of parsing it again. That allow us a better performance because it's faster and has a better availability. The problem is that the schedule can change during the year so we allow the student to force a new web request with a new parsing in order to update it (with a refresh button on the page).

4.4.5 University canteen

We choose to store the main informations(name, place, image, opening time) about restaurants in the database because it's something that will not change every year and there is not a lot of field to update if some modification are needed. We encode the 6 restaurants present on the

ucl website[2]. For the database we created a new table and encoded data manually. About the menu we couldn't store it in the database because of the existence of the daily menu (we should omit them or update them manually each week). We wanted to do some web parsing but the html code for each restaurant had his own syntax(even if they have the same rendering) ant thus we would need to create one parsing technique by university canteen which was a too heavy workload for the time we had. Instead we chose to create a button linking to the related menu page.

4.4.6 Sports

We couldn't store the sports in a database because the planning undergoes changes every year and we don't have manpower to update it. The sports department don't have web services either so the only solution we had was to parse the website. On a positive note, the Louvain-la-Neuve and Woluwe sites share the same website and thus the parsing is effective for both.

URL

The base url for the two sites is the same [3]. There exist two main query parameters possible

- The campus: A number defining the campus for which we want to show the sports schedule(1 for all, 2 for Louvain-la-Neuve, 4 for Woluwe). The factory create a specific request with 2(resp.4) when the selected campus is Louvain-la-Neuve(Woluwe)
- The skip: The sport web site response contains only 50 sports time slots. The skip argument permit to access the other sports instances after the first fifty.

It is a bit more difficult and slower than the other request because in this case we need to create a request by 50 sports time slots(for example if we have 132 sports we need three request. The first taking 0-50 instances, the second 51-100 and 101-132 for the last). Moreover the website is not 100% available some times, we couldn't access it because it was off line so we added a time-out method in order to prevent the user that the website is probably off-line.

Parsing

For the parsing the sport has quite close structure than with the schedule except different tag name so we used an other string parsing method based on regular expression. The sport website create a page that contain the sport closer than the day and the time we are. For example if we Friday 14 hours, it will show the Friday sports time slots after 14 hours first. Thus we iterate over all the time slots until we are sure that we capture all the sport for a week. We stocked the result in an other list of JavaScript object that we store and send to the front end.

[2]https://www.uclouvain.be/restaurants-universitaires.html

[3]`http://ucl-fms01.sipr.ucl.ac.be:82/ucl_sport/recordlist.php?`

Local storage

Exactly the same reasoning than for the schedule. Here we saved the result for both Louvain-la-Neuve and Woluwe in different table so we have them both in memory once they has been accessed at least once. Moreover each time the page is requested the software will reorganize the list in order to have the closer time slots first which are in general more suitable for the user.

Chapter 5

The application

In this section we will present the application as we implemented it. In a first time we will explain the design. In the second time, we will explain how it work from a technical point of view. All the page screen described in this section can be found in the annexe X.

5.1 The application UCLCampus

5.1.1 Header

design We create a global header for the application. The header has different color depending the section where we are(chap 4.2.3 for more details). Otherwise, it has three component, the first is the name of the page we are using(for example "Sainte-Barbe" if we are looking the detail of this one). Two buttons, the first is the back button, it allow you to back to the previous page. The second button open the settings menu. This header is by default enable on all the page of the application but the programmer can overwrite it if he want to add something on it(new buttons, search bar,... for example).

Implementation Ionic framework provide a file index.html in its architecture. This file has two purposes. The first is to list all the JavaScript present in the application in order to run them. The second is to create a user interface that will be everywhere in the application but that can be overwrite. We used it to create the header, the footer and the settings menu. It's a good point for the maintainability because here we have only code common for all the header except if they have been overwrite(thus it can be modify quickly and efficiently). I will not describe precisely how we implemented the header because it's just some html tags and css components that ionic framework provide

5.1.2 Footer

design The footer is a bar with buttons bellow the screen. As the header, the footer color change depending the section. There is four buttons on the footer, the four main sections(student, campus, tools, city). We though it was more user friendly to do that because you don't need to

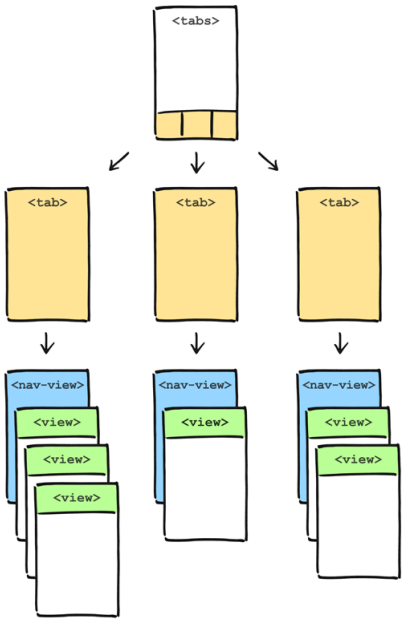


Figure 5.1: Tabs diagram

click on previous page until you reach the main menu if you want to explore an other section. For example if you are looking at the detail of a lecture hall and you want to see the events of the week, without footer you need to click three times on back button then select campus and events it is five manipulations. With footer, you can click on the campus button and then event, it is only two steps. The footer point out in which section you are, it will greyed out all the button except the one representing that one.

Implementation Ionic provide all the components and an extended architecture for the tabs(buttons in the footer). When we create a page, we create a state associated to it(see chapter 4.1.5). In this state, we can set an argument specifying the section it belongs. Once it is done, Ionic manage automatically the tabs function. For more details[1] see figure 5.1, it create a tree structure of states(called tabs) where each state hold a list of view(pages)and his own navigation history stack (important for back button). For example, if we are on the libraries page, there exist a state associate to it, informing that this page belongs to the student state and thus it manage automatically to highlight the student button in the footer in order to show the user that he is in the student section.

5.1.3 The homepage

When the user launch UCLCampus on his smartphone, he arrive on the homepage. This page contains the header and four buttons(student, campus, tools, city). The purpose of this page is to not force the user to be in a menu once he open the application. An other choice could be to put in the student section and let him use the footer button if he want to change, we

[1] `http://ionicframework.com/docs/api/directive/ionNavView/`

found this solution a bit sad and so we choose to create the homepage. We remove footer from the homepage because it's redundant to have the sub menu button at the same time as the content of the page and in the footer. I will not give technical detail for this part because there is nothing interesting, except that we created some css for the button.

5.1.4 The student/campus/tools/city menus

This four sub menus have the same design and work the same way. For the design, they display a list of buttons where each button is composed with the name of its functionality and an picture representing it.
TODO lien vers l'annexe des screens ou alors importer l'image.

design We chose a sleek style for each item of the list with just the name of the functionality and an icon representing it on the left. We choose to remove the back button of the header because it was useless to go back to the main menu(the user can use the footer in order to chose a new section).

Implementation The four menu have the same code skeleton. The principle is simple, each menu has his own factory. This factory hold a list where item represents a button. This items have different parameters:

- title: The name of the functionality the button is pointing to.
- icon: An icon representing the functionality
- url: The technical link to the new page. If this is set, website should be null
- website: The application can open an external website, in this case this field is the url. If this field is set, url should be null
- campus: A list of campus where the functionality is implemented. For example if we set it to ['Louvain-la-Neuve','Woluwe']. If the user's selected campus is one of this, he will see the button in the menu, if not the button will not be present. The list can be empty([]), this case is equivalent to write all the campus (more efficient for the programmer).

Then this list is simply pass to html and JavaScript controller that use it in order to create the user interface.

5.1.5 Studies

The student menu contains all information relevant to the academic year. All the functionality except libraries and lecture halls are available for every campus. We have libraries and lecture halls informations in database only for Louvain-la-Neuve and Woluwe.

Schedule

For the schedule we didn't want to implement a fresh new calendar algorithm with a lot of functionalities because it's already present on all smartphone as native agenda. So the purpose of this section was allow the student to export their ADE schedule directly to their agenda. Nevertheless we implemented a display function where the student can see the lectures he has for a selected day. In order to implement it we used the data send by the back end (explained in chap 4.4), we display this data as list with

- Time slot
- Course code - Professor
- Local

To only display the lecture for a selected day we create a JavaScript filter. In order to let the user select a day we used the onezone datepicker plugin(open source, MIT license[2]), it's just a date picker object that display a calendar where you can select the day you want. More, we implemented swipe right and swipe left functions in order to add or remove a day from the actual date. To export the list of lectures to the native calendar, we used the PhoneGap Calendar plugin [3](MIT). It allow us to create, remove and update events in the agenda. However this plugin is still in development and could create some issues, it should not be delivered to end users but we could not find a better solution. We add a button in the header to refresh the schedule if the student need it.

Lecture Halls

We implemented it in two phases. The first an overview of all the halls. For this the controller call a first time the factory method to have all the lecture halls in a list where we can access details efficiently (for example lectureHall.addr,...), in order to display them in the html. This menu use the general header and footer, and just display a list of auditoriums with

- A picture of it on the left side
- The name
- The address

If a user has an interest for a specific lecture hall, he can click on it in order to see its details. The details are a bigger image in order to have a better representation of the lecture halls, the address and a button linking to the map with the position and gps preconfigure for it. From a technical point of view, we call a first time the back end to have the list of all lecture halls and once the user select one, we redirect to a new url with the id as query parameter and we ask the factory again but only for the selected hall details.

[2] `https://market.ionic.io/plugins/onezone-datepicker`

[3] `https://github.com/EddyVerbruggen/Calendar-PhoneGap-Plugin`

Libraries

We used the same process than for the lecture halls except for one detail. The libraries have opening time, we found that display directly in the list if the library is closed or open a cool feature. We add it under the address as a small green section with the word open if it is or a red section with closed otherwise. If the user want to see the opening time, we add it in the detail. Technically, it was just the same process than for lecture hall with small modifications in order to include the opening times. For this, we add a JavaScript function that takes as parameter the opening time and return true or false.

Uclouvain.be and Moodle

Uclouvain.be and moodle are just link to the website in a first time, we think that it is possible to parse both but the Moodle website is already smartphone responsive and has a good display. The website uclouvain.be is too big to parse all informations, so we put a link in our application if the user want to search for specific informations. We choose the browser icon for the button with the aim to inform the user that it open a website and not a functionality we implemented.

Implementation

As explained in chapter 5.1.4 about the sub menus, we set the variable website for uclouvain.be and Moodle. Due to that when the user click on the button we call a function openUrl that we created. In order to open external URLs in Ionic we used the InAppBrowser plugin [4] which allow you to open it in application browser, system browser or Cordova webview. The plugin provide a navigation bar but this one has a poor design, so we choose to remove it for android because the user can go back to the application with the physical back button. However, we had to keep it for ios because there is the only way to have a way back to the application.

5.1.6 Campus

The campus section contains elements relative to live on the campus and that haven't direct link to the academic year.

Events

The events section is only available for the Louvain-la-Neuve campus. All the events informations (including pictures) come from the website `http://louvainfo.be/`.

design The events page can be break down in two parts. The first page with the list of all events sorted by starting date and the second part when you click an event in order to obtain its details.

[4] `http://cordova.apache.org/docs/en/3.0.0/cordova/inappbrowser/inappbrowser.html`

Events list The events list keep the classical header and footer. Each item of the list represent a event with three informations:

- The name of the event
- The starting date (date + time)
- The place

Moreover, each event has a category. We create a sub header where the user can open a combobox and select a specific category. Once a category is selected, the list is filter and shows only the relevant events. When a user click on an event in the list, it will open the page details.

Event details This page has the classic header and footer, the title in the header is the event name. I will describe the element in order of appearance from top to bottom.

- A picture of the event
- Start date and time
- End date and time
- The place
- The description of the event
- A first button opening the event page on `louvainfo.be`
- A second button opening the place on the map

We faced two problem with the description The first is that the RSS only send a text with a max length fixed, thus sometime we are no able to get all the description. In this case, we detect and add "..." to the end of the text so the user now the description is not complete and they can look at it on the website. Second problem, the description is a huge text and it take a lot of place on screen(the user need to scroll down a lot to access the button, it is not user friendly at all). To face it, we created an expandable text. At the beginning we only show the first two lines to the user and bellow then we add a button "⊕ more" once the user click on this button, the full text appear with at the end a new button "⊖ less"(his button reduce the text to the first two line). This allow us to show the user the description and the two buttons bellow it without a scroll need.

Implementation For the implementation, it work the same way than for the libraries and lectures halls. The factory find and parse all the data in list(of JavaScript object) that is sent to the JavaScript controller. The controller prepare data for the HTML which display it in order to create the user interface. Same for the details, the factory send the details as a JavaScript object to the controller that send it to the HTML. The list of category that the user can select

is create automatically by the application, it look at all event and their category and create a list with them.

University restaurants

As explained in chapter 4.4.5 we only include the restaurants listed on `https://www.uclouvain.be/restaurants-universitaires.html`. Thus this functionality is only operational on the Louvain-la-Neuve, Woluwe and Mons campus.

design The design of the restaurants is really close to the libraries, we first have a list of all restaurants in the campus with the following informations.

- A picture of the restaurant on the left
- The restaurant name
- The place
- Label "open" or "close"
- Opening on lunch time
- If exist, evening opening

The opening time have always the same format: date - date, time - time (for example Monday to Saturday, from 12h to 14h) The user can select a restaurant in order to see the details. The details are in order of appearance from top to bottom (the restaurant name is the header title).

- A picture of the restaurant
- Noon time opening
- If exist evening opening time
- Address
- Description
- Button "See on map"
- Button "Menu"

For the description, we used the same way for events and restaurants(an expandable text, see more details in the events section). The button "Menu" open the specific menu url for the restaurant. The button see on map open the map functionality with the restaurant preselect in order to start the gps guide.

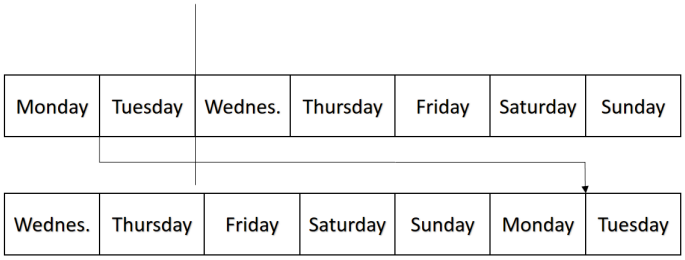


Figure 5.2: Sorting day operation

Implementation The factory retrieve all the database informations to the restaurant controller(js). The controller send the data to the html that is seen by users. We couldn't reuse the javaScript code that detect if a library is open or close because the restaurant could possibly have an evening opening, so we created a new JavaScript file with all utility functions and place it in the folder for restaurants.

Sports

The sport section is available for two campus: Louvain-la-Neuve and Woluwe.

design There is only one page for the sport. First, we introduce a sub header where the user can select to see only one specific sport or the global schedule with all available sports. Under header, the sports are display into a list. Each sport in the list contain the following details:

- The name of the sport (for example Badminton). If the user select a specific sport, this information become useless, in this case we remove this field.
- Day and time (ex: Saturday 20:00 - 21:30)
- Place

The list is ordered by date and time. The first sport instances in the list are the one of the day. Then it follow a classic order. For example if we look at the sport list a Wednesday, the first sport time slot we will see are those of the day, then those of Thursday and so on.

Implementation The factory retrieve data from website or from database if it has already been parsed. The special trick is that we want the factory to return a list of sport ordered by day starting from the day we are. The UCL sport website already order the sport by date, so we just kept their order while parsing. When the user connect a new day, we will pop the block of sport of the day before it and put them at the end of the list as show in figure 5.2. The list of sport type(that the user can select in the combo box) is created in the controller. This one automatically update and display sport in alphabetical order. Then the controller communicate all the sport time slots and the list of distinct sports to the HTML that is seen by the user.

5.1.7 Tools

The actual only functionality of tools are lln maps [5] . We created the menu in order to introduce new functionalities later.

LLN Maps

Design lln maps has its own header and footer so we discuss with our contributors about the way to merge them to our application. First, they have a search side menu so we decide to overwrite our settings side menu that is unreachable from the maps. Therefore, the settings button in the header is replaced by a search button. In their footer, they have a compass or map mode. Actually, we didn't fix a choice, There exist a lot of alternatives.

- Put the compass button in the header
- Add it in our footer
- Create a sub footer
- Create a sub header
- Put it on the map page (not in a footer or header)

Implementation We do not pretend to discuss about the implementation of lln maps that is not the purpose of this section. We will discuss the technical points useful for our application. The main interactions with lln maps are the buttons "See on maps" placed throughout UCLCampus. They inform us that we could use url in order to open a preselected place and have the gps ready to be launched. Though lln maps is not fully integrated and thus our button not linked yet, we saw that some abbreviations of lln maps and UCLCampus (for lecture halls,...) are different so we created a database table in order to gather them.

5.1.8 City

For the city section, we only created the menu in order to add new functionalities later. However, this section is more sensitive than other because it add external business in the application and it is a kind of advertising, so more advanced discussion should take place when adding a functionality.

5.1.9 Others

Settings menu

The settings menu can be open everywhere in the application except for the map.

[5] `https://github.com/mathieuzen/llnmaps`

Design This menu is a side menu, that mean when the user open it, it does not take all the screen as a new page. Instead it cut the screen in two parts (cut vertically). The part on the left is the page where the user opened the settings menu, the second part is the settings menu. That allows the user to change a setting without being perturbed in its actual task. The settings menu offers three items:

- The language
- The campus
- Logout

Language We introduced three different languages, French, Dutch and English. The student can select one of this three languages and all the application will directly been translated into it. Actually the translation in Dutch is not done (though it is implemented).
For the implementation, we used a plugin called "angular-translate" [6](MIT). This plugin allows to create a configuration where you define the application languages, and for each languages it hold a kind of dictionary with the variable and his translation.
Here is an example of the code for translation:

```
$translateProvider.translations('en', {
    hello_message: "Hello",
    goodbye_message: "Goodbye"
});
```

In this example, we create an English translation and the variable "hello_message" is translated by "Hello" in the user interface (same for goodbye). Our application has more variables to translate but the process is exactly the same. One more step is needed to translate the variable, at each time we want a variable in the html to be translated we need to apply a translation filter onto it. For example

```
<html tag>{{"hello_message" | translate}}</html tag>
```

The plugin allows to set a preferred language that will be preselected at the application launch. Actually it is English.

Campus selection The user can select a campus with the button named "Choose my campus" in the settings menu. This button open a page with the list of all possible campus, the user has to choose one and validate his choice. As explained earlier the campus selection influences the available functionalities in application. A pre selection based on the geolocation happens when the user open UCLCampus and the nearest campus is chosen per default.
For the implementation, we created a campus factory that hold a list of campus. The items of

[6] https://github.com/angular-translate/angular-translate

this list have three attributes, the name of the campus, longitude and latitude. In order to select per default the closest campus, we created a JavaScript function that just return the campus minimising the distance between it and the actual position of the user. Thus it is easy to add a new campus to the application or remove one, but the programmers should care about the compatibility between the new campus and all the functionalities enable for all sites.

5.2 Modularity and how to add a new functionality

This application was built with the idea that other contributors would have to add functionalities themselves or even take over the project in the future. It was therefore important to be able to add modules to the application without having to understand the whole code.

This section will present a user manual on how to get the application to run from scratch and how to add new functionalities. It must be noted that in order to deploy the application on iOS, one must user OS X as his operating system. The following steps will however work for any operating system and the user will be able to deploy the application on Android regardless of the OS used.

5.2.1 Setting up the framework

Step 1: Installing Git

First, we need to install Git.

For Windows users:
Go to https://git-scm.com/download/win, download and install Git for Windows. Any further commands in this guide can be executed in either the Git Bash.

For Linux users:
Type the following line of code into your terminal:

```
1  $ sudo apt-get install git
```

Step 2: Installing Node.js

Node.js is a JavaScript runtime that is required to install the other things one needs to install the application. This step slightly differs depending on your OS.

For Windows users:
Windows users can download an installer from the Node.js website: https://nodejs.org/en/ .

For Linux users:
First, install Node.js by typing the following command line in your terminal:

```
1  $ sudo apt-get install nodejs
```

Then, install npm:

```
$ sudo apt-get install npm
```

Finally, create a symbolic link for node:

```
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

You can now test that the installation worked properly by running these two commands:

```
$ node -v
v0.10.25
$ npm -v
1.3.10
```

Step 3: Installing Ionic and its dependencies

If you are on Windows, drop sudo from the following commands.
First, install Cordova:

```
$ sudo npm install -g cordova
```

Then, install Ionic:

```
$ sudo npm install -g ionic
```

5.2.2 Installing UCLCampus

Now that you've set everything up, it's time to fetch the source code of the application from GitHub.

Step 1: Cloning the project

First, open a terminal into the repository of your choice. Then, enter the following command:

```
$ git clone https://github.com/amoyaux/UCLCampus.git
```

Step 2: Installing the required packages

Now you will need to install some packages before running the application. To do so, move to the right directory:

```
$ cd UCLCampus/UCLCampus/
```

Then run the following commands. These commands, except for the first one, can be found in the plugin.txt file.

```
$ npm install
$ cordova plugin add cordova-plugin-device
$ cordova plugin add cordova-plugin-console
$ cordova plugin add cordova-plugin-whitelist
$ cordova plugin add org.apache.cordova.network-information
```

```
$ cordova plugin add https://github.com/brodysoft/Cordova-SQLitePlugin.git
$ cordova plugin add https://github.com/litehelpers/Cordova-sqlite-storage.git
$ cordova plugin add https://github.com/EddyVerbruggen/Calendar-PhoneGap-Plugin.git
$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-inappbrowser.git
$ cordova plugin add cordova-plugin-geolocation
$ cordova plugin add https://github.com/brodysoft/Cordova-SQLitePlugin.git
$ cordova plugin add https://github.com/an-rahulpandey/cordova-plugin-dbcopy.git
$ cordova plugin add cordova-plugin-network-information
```

5.2.3 Running the application

There are several ways to run the application now that it is installed. Note that in order to run it on emulators, you will need to install said emulators and their required dependencies (programming languages, ...). This aspect is not covered in this guide.

Running the application in an Internet browser

Since Ionic is an hybrid framework, the application can easily be run in a browser. Simply open a terminal in the UCLCampus/UCLCampus/ folder and run the following command:

```
$ ionic serve
```

The application is now running on http://localhost:8100/ . However, some functionalities may not work properly, namely the ones that need access to the SQLite database. This is due to the fact that opening the database is not done in the same way for a browser or a mobile device. It would be possible to add support for the database in this case, but this was not a priority as the focus of the application is not to be run in a browser.

5.2.4 Running the application on a device or emulator

Once you have installed the emulator or connected the device to your computer, you need to add the corresponding platform to Ionic using the following command:

```
$ ionic platform add android
```

or

```
$ ionic platform add ios
```

Note that to run the application on an IOS device, you will need a Mac computer. Now that the device is connected and the platform is added, simply run the following the run the application:

```
$ ionic run android
```

or

```
$ ionic run ios
```

For more options on the launch command, please refer to http://ionicframework.com/docs/cli/run.html.

5.2.5 Adding a new functionality

Now that everything is up and running, new functionalities can be added. This guide will explain how to add a new functionality to the Tools section. The same steps can be repeated to add new functionalities to any section of the application.

Step 1: Creating the required files

First, you will need to create the required files for your functionality. Navigate to the UCLCampus/UCLCampus/www folder. In this folder, we can see the architecture of the project. It contains one folder for each section of the application. Now, since we want to add a functionality to the Tools section, navigate to the tools folder. Here you can see 3 folders: ”js”, ”templates” and ”maps”. ”js” and ”templates” are the folders that handle the tools menu in the application while ”maps” is a functionality. So to create a new functionality, you will need to create a new folder.

Create a new folder named ”my_functionality”. Now navigate to this folder and create two new folders, ”js” and ”templates”. ”js” will contain your JavaScript while templates will contain your html code. In the ”js” folder, create a ”my_functionality.js” file and create a ”my_functionality.html” file in the ”templates” folder.

Step 2: Writing the JavaScript code

Open your favorite text editor and open the ”my_functionality.js” file. In this file, paste the following code:

```
angular.module('ionicApp').controller("MyFunctionalityController", function(
    $scope) {
   $scope.textToDisplay = "This is the text we want to display."
});
```

Step 3: Writing the HTML code

Open the ”my_functionality.html” file. In this file, paste the following code:

```
<ion-view title="Title">
   <ion-content>
      <div>
         {textToDisplay}
      </div>
   </ion-content>
</ion-view>
```

Step 4: Adding your functionality to the application

To add the functionality to the application you will first need to create a state for it. To do so, open the UCLCampus/UCLCampus/www/js/app.js file. In this file, you will find the state provider. It contains a list of all the states the application can be in. You need to add a state for your functionality. Find this state in the list:

```
.state('app.tools', {
    url: "/tools",
    cache : false,
    views: {
      'tools-tab' :{
        templateUrl: "tools/templates/tools.html",
        controller: "ToolsController"
      }
    }
  })
```

This state corresponds to the tools menu. Below this state, add the following new state:

```
.state('app.myfunctionality', {
    url: "/myfunctionality",
    views: {
      'tools-tab' :{
        templateUrl: "tools/my\_functionality/templates/my\_functionality.html",
        controller: "MyFunctionalityController"
      }
    }
  })
```

Once this is done, open the UCLCampus/UCLCampus/www/index.html file. In this file, find the following lines:

```
  <!-- tools module -->
  <script src="tools/js/tools.js"></script>
```

Below these lines, add the following:

```
  <script src="tools/my_functionality/js/my_functionality.js"></script>
```

Your functionality has now been added to the application ! However, we don't have a way to access it yet. We will therefore add a button in the tools menu to access it. Open the UCLCampus/UCLCampus/www/js/factory.js file. This file handles, among other things, the buttons we can see in the different menus of the application. Find the function called "ToolsMenuFactory". In this function you can see the following code:

```
  toolsMenuList : [
      { title: 'Maps' , icon:'icon ion-map royal', url:'app.maps', campus:['
          Louvain-la-Neuve']}
  ],
```

Change it to the following:

```
1   toolsMenuList : [
2      { title: 'Maps' , icon:'icon ion-map royal', url:'app.maps', campus:['
           Louvain-la-Neuve']},
3      { title: 'My functionality' , icon:'icon ion-help royal', url:'app.
           myfunctionality', campus:[]}
4    ],
```

The campus field in each item is used if you want to create functionalities specific to one or several, but not all, campuses. For instance, the maps functionality is currently only available for the Louvain-La-Neuve campus. An empty campus field means that we can access the function from all campuses.

You can now access the new functionality from the tools menu of the application. You can now start editing the JavaScript and HTML files to create a more complicated function.

5.3 Future functionalities and possible improvements

Chapter 6

Analysis

Here we will reflect about the many choices we made and try to decide wether they were the right ones or not.

6.1 Ionic framework

6.2 GitHub

6.3 Project Management

Chapter 7

Conclusion

Chapter 8

Bibliography