
UCL

**Université
catholique
de Louvain**

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
options:

by : Arnold MOYAUX

Baptiste LACASSE

Louvain-la-Neuve
Academic year 2015-2016

Contents

1	Introduction	4
2	Background	5
2.1	Cross-platform mobile development tools	5
2.1.1	The native approach	5
2.1.2	The web approach	6
2.1.3	The hybrid approach	6
2.1.4	Our choice	6
2.2	Open-source project and code sharing	6
2.3	Project Management Methodologies	7
3	Functionalities of UCLCampus	8
3.1	Choice of functionalities and sections	8
3.2	User interface	11
4	Implementation	13
4.1	Architecture	13
4.1.1	Folder organisation	13
4.1.2	Information processing	14
4.1.3	Front-end and back-end	15
4.1.4	Factory	15
4.1.5	State provider	16
4.2	Coding standards	17
4.2.1	Header and footer	17
4.2.2	Tree hierarchy	17
4.2.3	Colors	17
4.3	Security	18
4.4	Information retrieval	18
5	The application	19
5.1	The application UCLCampus	19
5.1.1	Studies	19

5.1.2	Campus	19
5.1.3	City	19
5.1.4	Tools	19
5.1.5	Others	19
5.2	Modularity and how to add a new functionality	19
5.3	Future functionalities and possible improvements	19
6	Analysis	20
6.1	Ionic framework	20
6.2	GitHub	20
6.3	Project Management	20
7	Conclusion	21
8	Bibliography	22

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. // TODO

2.1 Cross-platform mobile development tools

In each of these sections, we will detail the different approaches one could choose to develop a cross-platform mobile applications. We will also present several frameworks using these approaches. We will then compare them and choose one of those approaches for the rest of the project. //TODO

2.1.1 The native approach

The first approach we considered for our project was what we call a native approach. The native approach consists in using the native technology and language for each platform, for instance Java for Android and Objective-C for iOS.

Pros	Cons
Best achievable performance	Low maintainability
Always up-to-date with the latest API	Harder to find contributors fluent in all technologies
Can use any platform	Can lead to different versions of the application

Table 2.1: Pros and cons of the native approach

2.1.2 The web approach

A second approach we considered was the web approach. This approach consists in using HTML5 to develop an application that will be usable on any platform.

Pros	Cons
Can be used on any mobile platform	Doesn't have access to native platform features
Easy to find contributors fluent in HTML5	Harder to implement local storage/security (//TODO NEED BETTER SOURCE)
Easy to maintain	Not as performant as native

Table 2.2: Pros and cons of the web approach

2.1.3 The hybrid approach

The last approach to develop a mobile application is called the hybrid approach. An hybrid app is mostly built using HTML5 and JavaScript and is then wrapped inside a thin native container, giving it access to native features.

Pros	Cons
Can be used on any mobile platform	Not as performant as native
Easy to find contributors fluent in HTML5 and JavaScript	
Easy to maintain	

Table 2.3: Pros and cons of the hybrid approach

2.1.4 Our choice

2.2 Open-source project and code sharing

In this section, we will explain the choices we made concerning the code sharing platforms we used as well as the licenses we used to protect our work.

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.

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.

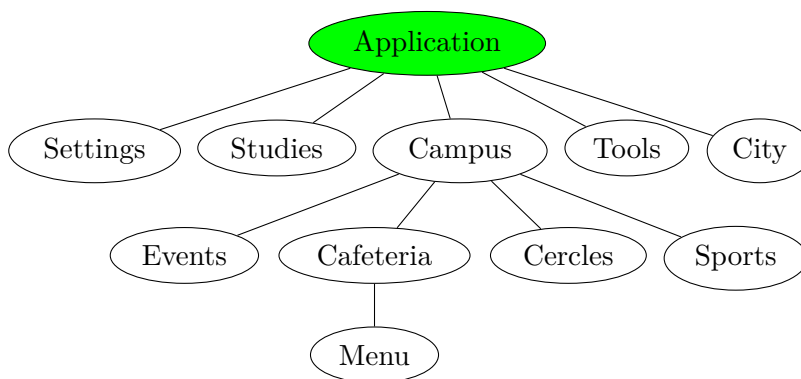
Chapter 4

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 can add its functionalities at each level to the application. For this we thought our implementation as a tree. At the top level we have the application with the settings menu and base configuration after 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.



Each node having his own JavaScript,html and css (if needed) code.

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(maybe having more than one js and html). 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,

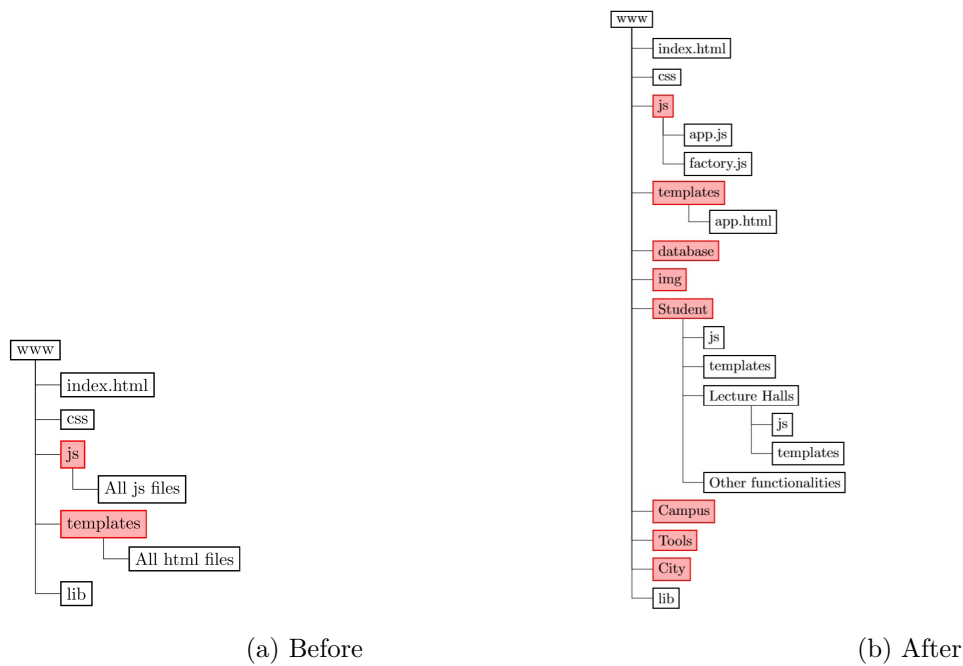


Figure 4.1: Folder evolution

you have directly access to the related files. Figure 4 1 is a summary of the change, red folder are those we modify.

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

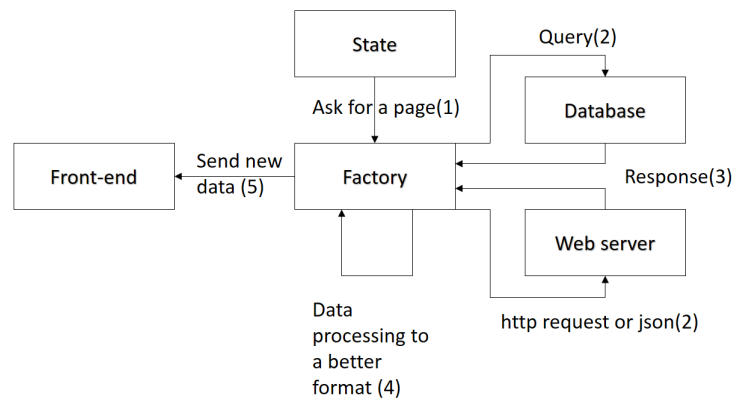


Figure 4.2: Factory operation

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 configuration of a page (if it's 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.

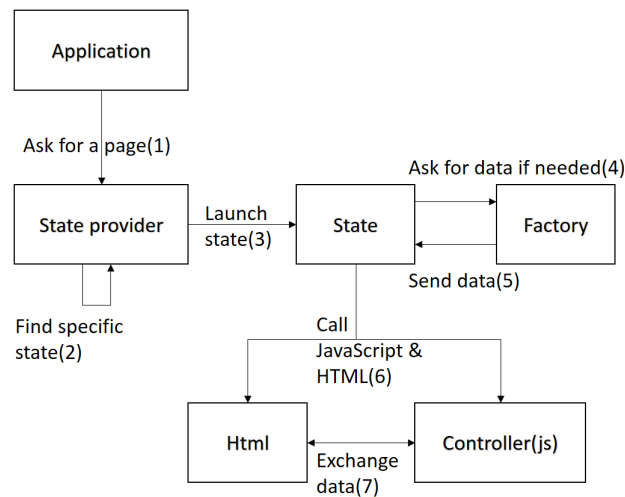


Figure 4.3: State provider operation

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

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

. 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.

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 chose to not impose any rules as long as it stay in the relative module.

4.2.3 Colors

We used the colors code of the UCL¹. For this we modified the sass theme of Ionic so module developed with basic ionic css component should update to 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

4.4 Information retrieval

Chapter 5

The application

In this section we will present the application as we implemented it.

5.1 The application UCLCampus

5.1.1 Studies

5.1.2 Campus

5.1.3 City

5.1.4 Tools

5.1.5 Others

5.2 Modularity and how to add a new functionality

5.3 Future functionalities and possible improvements

Chapter 6

Analysis

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

6.1 Ionic framework

6.2 GitHub

6.3 Project Management

Chapter 7

Conclusion

Chapter 8

Bibliography