

UCLCampus

A mobile application for UCL students

Dissertation presented by
Baptise LACASSE , Arnold MOYAUX

for obtaining the Master's degree in
Computer science

Supervisor(s)
Yves DEVILLE

Reader(s)
Kim MENS, Hildeberto MENDOÇA , Mathieu ZEN, Jorge PEREZ MEDINA

Academic year 2015-2016

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 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 | 7 |
| 2.1.5 | Choosing the best hybrid framework | 8 |
| 2.2 | Open-source project and code sharing | 8 |
| 2.2.1 | Open Source And Free Software | 8 |
| 2.2.2 | License | 9 |
| 2.2.3 | GitHub | 10 |
| 2.3 | Existing Student Application | 11 |
| 2.4 | Project Management Methodologies | 12 |
| 2.4.1 | Agile Development | 12 |
| 2.4.2 | The Project Step By Step | 12 |
| 2.4.3 | ZenHub | 13 |
| 3 | Functionalities of UCLCampus | 16 |
| 3.1 | Choice of functionalities and sections | 16 |
| 3.2 | User interface | 19 |
| 4 | Implementation | 21 |
| 4.1 | Architecture | 21 |
| 4.1.1 | Folder organisation | 21 |
| 4.1.2 | Information processing | 21 |
| 4.1.3 | Front-end and back-end | 23 |
| 4.1.4 | Factory | 23 |
| 4.1.5 | State provider | 24 |
| 4.2 | Coding standards | 25 |
| 4.2.1 | Header and footer | 25 |
| 4.2.2 | Tree hierarchy | 25 |
| 4.2.3 | Colors | 25 |
| 4.3 | Security | 25 |
| 4.4 | Information retrieval | 26 |
| 4.4.1 | Student data(OSIS) | 26 |
| 4.4.2 | Libraries and lecture halls | 27 |
| 4.4.3 | Events | 28 |
| 4.4.4 | Schedule | 28 |
| 4.4.5 | University canteen | 30 |
| 4.4.6 | Sports | 30 |

| | | |
|----------|---|-----------|
| 5 | The Application | 32 |
| 5.1 | The application UCLCampus | 32 |
| 5.1.1 | Header | 32 |
| 5.1.2 | Footer | 32 |
| 5.1.3 | The Homepage | 33 |
| 5.1.4 | The Student/Campus/Tools/City Menus | 33 |
| 5.1.5 | Studies | 34 |
| 5.1.6 | Campus | 35 |
| 5.1.7 | Tools | 38 |
| 5.1.8 | City | 39 |
| 5.1.9 | Others | 39 |
| 5.2 | Modularity and how to add a new functionality | 40 |
| 5.2.1 | Setting up the framework | 40 |
| 5.2.2 | Installing UCLCampus | 41 |
| 5.2.3 | Running the application | 42 |
| 5.2.4 | Running the application on a device or emulator | 42 |
| 5.2.5 | Adding a new functionality | 43 |
| 5.2.6 | Step 5: submitting your functionality | 44 |
| 5.3 | Future functionalities and possible improvements | 45 |
| 5.3.1 | Future functionalities | 45 |
| 5.3.2 | Possible improvements of existing functionalities | 45 |
| 5.3.3 | Other improvements | 46 |
| 6 | Analysis | 47 |
| 6.1 | Ionic Framework | 47 |
| 6.1.1 | Benefits | 47 |
| 6.1.2 | Drawbacks | 47 |
| 6.2 | Testing | 48 |
| 6.2.1 | Unit Test | 48 |
| 6.2.2 | Performance testing | 48 |
| 7 | Conclusion | 51 |
| 8 | Bibliography | 52 |

Chapter 1

Introduction

The goal of this thesis was to develop an open-source, multi-platform mobile application for UCL students. The application would be used by students to help them with their everyday student life in a town they might not know much about. It would, for example, guide them to their lecture halls or libraries and show them the many activities the town has to offer. The application in itself, while useful, does not constitute this thesis.

The difference between a master's thesis and other assignments is that the goal of the thesis is not limited to evaluation and self-improvement. Indeed, a thesis should contribute to the field it is related to, providing insight for other researchers. For some thesis, the contribution is clear and does not need to be discussed any further. This was not the case for this thesis. Indeed, the title, "UCLCampus, a mobile application for UCL students", doesn't indicate what contribution we will be making.

We therefore defined two different contributions we wanted to make and decided to model this paper around them. Each of them will define a type of researcher or student that might be interested in reading this thesis.

The first type of readers contains people that are interested in developing another similar application. Indeed, the application we developed is open-source and works on several mobile platforms. Making an application with such attributes implies important design choices that are not always easy to make. The reader might therefore be interested in seeing the choices we made as well as our thought process. This information can be found in the second chapter of this thesis.

The sixth chapter of this thesis contains an analysis of the finished product and evaluates whether we made the right choices at the start of the project, which will also interest the first type of readers.

We hope that such readers will find this thesis useful but still invite them to conduct their own research before developing their application as the field is in constant evolution.

The second type of readers are the future students or researchers willing to contribute to the UCLCampus application. These readers will be mostly interested in chapters 3, 4 and 5. The third chapters explains the reasoning behind the functionalities we decided to add to the application. The fourth chapter depicts the way we implemented the application and explains the design choices we made. Finally, the fifth chapter presents the final product and explains how future users can contribute to the project while also giving them examples of possible future functionalities.

As the astute reader will already have noticed, the thesis follows the chronological order of the project, starting by detailing the background in chapter 2, detailing the project in itself in chapters 3, 4 and 5 and finishes with an analysis of the project as a whole in chapter 6.

We hope readers will find insightful informations in this paper and that it will help them in their endeavors.

Chapter 2

Background

The first thing to do when starting any new project is to choose the way you are going to work. These initial choices are not to be made lightly as they will influence the rest of the project and are often difficult, if not impossible, to revert. This section will present the different choices we had to make before starting our project and explain our thought process.

2.1 Cross-platform mobile development tools

The first challenge of this project was choosing the best technique to develop our application across multiple platforms. Indeed, the field has not matured yet and is constantly evolving, with trends appearing and disappearing constantly. It appears that no rigorous research has been done to compare these different trends and that even experts only support their claims with anecdotal evidence.

In this context, making an informed decision proved to be challenging. This section details the information we gathered and the choice we ended up making. We however encourage the reader to make his or her own research as the field might have evolved since this was written.

2.1.1 The native approach

The first approach we considered for our project was what is known as the 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.

Using the native environment means you will of course have access to every feature and achieve the best performance possible. However, this also means that you will need to maintain one version of your code for each platform you decide to support, thus decreasing maintainability and making harder for anyone to contribute to the project. Table 2.1 summarizes the pros and cons of this approach.

| 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.

Using a web page to display your application makes it supported on any platform but the additional layer results in overall poorer performances. This layer also makes native features unreachable meaning you might not be able to implement some functionalities. Table 2.2 details the different pros and cons of the web approach.

| Pros | Cons |
|---|---|
| Can be used on any mobile platform | Doesn't have access to native platform features |
| Easy to find contributors fluent in HTML5 | Not as performant as native |
| Easy to maintain | |

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.

The native container alleviates some the issues of the web approach but still reduces performances when compared to a native application. Table 2.3 shows the pros and cons of this final approach.

| Pros | Cons |
|--|-----------------------------|
| Can be used on any mobile platform | Not as performant as native |
| Easier to update | |
| 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

There isn't one approach that is inherently better than the other ones, the best one depends on what you are trying to accomplish.

The native approach is best suited when accessing native features of each platform is very important for the application. It is also the choice to be made if you can afford separate budgets for developers in iOS and Android.

The web approach is the best choice if your application is mostly a support for your main web application.

Finally, the hybrid approach is best used when you want quick results and need to make frequent updates to your application. It is also useful if you want to include outsiders to the project who aren't fluent in all native technologies.

In the context of this thesis, there is no budget therefore choosing the native approach will only depend on the importance of accessing native features. There also won't be a web application, meaning the web approach is probably a bad choice.

Furthermore, we believe that making it easier for outsiders to contribute is more important than accessing native features. Indeed, we don't think we will need any feature that would be supported by one platform but not the other and will mainly use generic features. Furthermore, the application will be updated every year (new lecture halls, libraries, ...). We therefore decided to go with an hybrid approach for this thesis.

2.1.5 Choosing the best hybrid framework

There are many different hybrid frameworks and choosing one was no easy task. We were however able to find the following comparison of the most popular frameworks (source <http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps/>)

| Framework | Native look and feel | Prerequisites | Community | Docs | Tools |
|---------------|----------------------|------------------|-----------|------|--|
| Ionic | 7/10 | AngularJS | 9/10 | 8/10 | Powerful CLI, Ionic SDK |
| Onsen UI | 6/10 | AngularJS | 4/10 | 9/10 | Monaca Could IDE |
| Framework 7 | 8/10 | HTML, CSS and JS | 6/10 | 8/10 | None |
| React Native | 8/10 | React | 8/10 | 5/10 | React Developer Tools extension for Chrome |
| jQuery Mobile | 3/10 | jQuery | 8/10 | 5/10 | None |
| Native Script | 8/10 | JavaScript | 5/10 | 9/10 | Free CLI, other paid options |
| Famous | 7/10 | WebGL, AngularJS | 3/10 | 5/10 | None |

This table seems to be the most complete comparison of hybrid frameworks existing at the moment and it shows that Ionic is the most balanced and that it doesn't have any prominent weaknesses. Despite the lack of objectivity of the measurements used, this table seemed to reflect the opinion of other authors comparing hybrid frameworks. Since realizing benchmarks for all those frameworks would represent a tremendous amount of work that would go well beyond the scope of this thesis, we decided to trust the advice of the aforementioned reviews. We thus decided to use the Ionic framework in the context of this thesis.

2.2 Open-source project and code sharing

2.2.1 Open Source And Free Software

What is an open source, free software?

A free software ensure three freedom:

- The usage freedom, commercial or not. This allows to use the software for any purpose.
- The modification freedom. This allows to modify the source code.
- Distribution freedom. The right to spread the software in every possible way.

An open source software want less strict criteria. For example an open source license can prevent a user to sell a source code or prohibit his modification. The source code for free software meet the open source criteria, not the opposite. A lot of software are free and open source at the same time.

Create an open source project

To create an open source software, you need two conditions:

- A project
- A community

This thesis has for purpose to create a solid foundation for the project. The project exist, it is the UCLCampus application and it is available on mobile. There is no community at this moment, Create a community around the project could be another thesis. A community doesn't mean only programmers, it can be translation student that can help or designer to create logo and icons.

2.2.2 License

The license are important in every project but even more in an open source project. Licenses restricts the use of the code by other programmers and companies. This section has three parts.

- Well know licenses in software development and their purpose.
- The license chosen for UCLCampus and why.
- The licenses of plugins involved in the application.

Most Common Licenses

The three licenses GPL(2.0 and 3.0),MIT and Apache represents 72% of the free software licenses¹

MIT The MIT license allows the right to every person having the source code to use, modify, merge, publish, sell it and change the license. The only restriction is to have the name of the authors with it and the copyright notice.

Apache The Apache software foundation allows to use their code the same way than MIT. It only forbid three points².

- "Redistribute any piece of Apache-originated software without proper attribution."
- "Use any marks owned by The Apache Software Foundation in any way that might state or imply that the Foundation endorses your distribution."
- "Use any marks owned by The Apache Software Foundation in any way that might state or imply that you created the Apache software in question."

The source code has to have a reference to the license instead of being written in every files.

GPL or GNU general public license. This license ensure the right of a free software. Moreover this license is copyleft. Copyleft mean that derived work have to be distributed under the same license. This ensure that the freedom will be preserved whenever the work is distributed.

The most used version is the 2 but there exist a new version 3. The two versions have the same goal but the version 3 add some conditions against companies that found tricks to cheat on the version 2³

¹<https://www.blackducksoftware.com/top-open-source-licenses>

²This three quotes come from their FAQ <http://www.apache.org/foundation/license-faq.html#WhatDoesItMEAN>

³Tivoization, Laws prohibiting, Discriminatory patent deals <http://www.gnu.org/licenses/quick-guide-gplv3.en.html>

| Plugins | License |
|---|-------------------|
| cordova-plugin-device | Apache 2.0 |
| cordova-plugin-console | Apache 2.0 |
| cordova-plugin-whitelist | Apache 2.0 |
| cordova-plugin-network-information | Apache 2.0 |
| Cordova/PhoneGap sqlite storage adapter | MIT or Apache 2.0 |
| PhoneGap Calendar plugin | MIT |
| Cordova-plugin-inappbrowser | Apache 2.0 |
| cordova-plugin-geolocation | Apache 2.0 |
| cordova-plugin-dbcopy | Apache 2.0 |
| Angular-translate | MIT |

Figure 2.1: Plugins and their associated license

UCLCampus License

UCLCampus has the license GNU GPL v3.0. These are the justification of this choice:

- This license ensure that derived work will keep this license. This prevent companies to take our code, add some features, add advertisements and make money on it.
- A lot of Ionic plugins use the license Apache 2.0. This license is only compatible with the version 3.0 of GPL.
- UCL can change it to a more strict license.

Plugins Licenses

Figure 2.1 show the list of the plugin and their licenses. There is a majority of Apache 2.0 and some MIT. About the compatibility between license, Apache 2.0 and MIT can both be used in GPL 3.0 project. The opposite is not true, GPL 3.0 can not be used in Apache 2.0 and MIT project. Apache 2.0 and GPL 2.0 are not compatible.

2.2.3 GitHub

GitHub is the state of the art for code sharing and project management on open source project. It have a lot of benefits:

- People can easily copy the code to their own computer and start a derived work.
- People can submit their local new features to the project owners as "pull request". The owner can accept or reject them.
- People can create issues if they found bugs or want new functionalities.
- There exist plugin to create a forum linked to the git repository. A forum is useful when other programmers has questions about your code or want to discuss new features.
- There are plugins to test or deploy automatically your project.
- Create a wiki for the repository. ⁴

The actual GitHub repository for the project is <https://github.com/amoyaux/UCLCampus>.

⁴Jenkins for example



Figure 2.2: LLN Campus home menu

2.3 Existing Student Application

The number of university application on Google Play is huge. They often offer a schedule, libraries, lecture halls and maps features. However some application are more interesting in the UCLCampus case because they are close or can be directly related to our project.

LLN Campus

LLN Campus was a project developed by 3rd years UCL computer sciences students. This project aims to help the the UCL student in Louvain-la-Neuve. This project is only available for android and is developed in Java. It contains different menus, schedule, libraries, lecture halls, hobbies, map and links to useful student websites (see figure 2.2). They implemented useful features as an indicator for the libraries showing if they are open or not. Their project is available on GitHub. They created a database for Louvain-la-Neuve that we reused in our application.

LLN Maps

LLN Maps developed to the LSM, is an application based on maps. It is implemented with Ionic. It contains various informations as lecture halls, libraries or train station. This informations are available as markers on a map. A gps guidance with time calculation is possible from the actual position to each points of interest. There is a search bar if a specific information is required. Moreover a compass mode is possible. See figure 2.3 for a visual. LLN Maps will be integrated in UCLCampus.

Quivr

Quivr ⁵ is the KUL application. It is not open source. This application is developed by 22 members and have 60 000 users. It has a lot of features and it takes into account all KUL sites, not only Leuven. The different features are a schedule, a friends system, libraries, university restaurants and news. They have really impressive features as a system that tell which library is open and how many places are used. An other feature is to share schedule with friends in order to meet up easily.

⁵<https://quivr.be/home/>

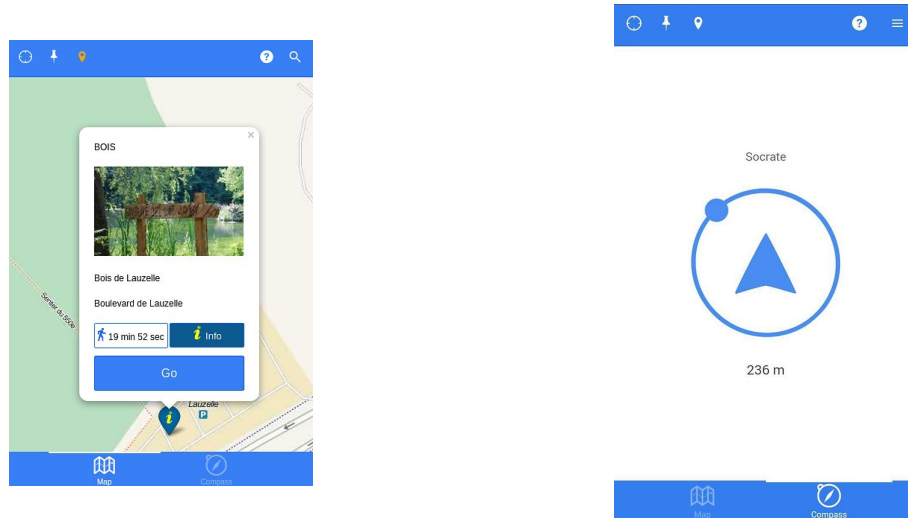


Figure 2.3: LLN maps

2.4 Project Management Methodologies

Here we detail the choices we made as to how we were going to manage the different parts of the project. We worked on an agile approach with a sprint system.

2.4.1 Agile Development

An agile method has a good compatibility with the project. Agile development is based on different principles, the most interesting in our case are:

- A fast and continuous delivery to the customer, fast feedback.
- Flexible requirements, even in development.
- Weeks delivery rather than months.

2.4.2 The Project Step By Step

Firstly, the steps required before the project implementation:

1. Define the functionalities of the application. This functionalities are defined under the form of user stories. Example, "As a student, I want to be able to export my ADE schedule into my agenda". The details of this step are in section 3.1.
2. Create the application frames. An application preview allow us to discuss the functionalities and the design with people that are not related to the project. The details are in section 3.2.
3. Think about an architecture. The code should be easy to maintain, learn and handle. The architecture is explained in chapter 4.1
4. Create tasks from user stories. Each task has complexity points. Often, more the complexity is high, more time it takes.
5. Initialise the tasks board.

During the development, the process is :

Activity on campus #20

Closed amoyaux opened this issue on 8 Dec 2015 · 0 comments

Edit

amoyaux commented on 8 Dec 2015

Owner +😊 ✎

Labels: None yet

Milestone: No milestone

Estimate: 13

Assignees: amoyaux

Epics: Not inside an Epic

amoyaux modified the milestone: **S1** on 8 Dec 2015

amoyaux changed the pipeline from **Campus** to **To Do** on 16 Mar

amoyaux changed the pipeline from **To Do** to **In Progress** on 16 Mar

Figure 2.4: Issue details

1. Create a new milestone. A milestone define a date for which a set of tasks should be done.
2. Select the tasks to do for this milestone.
3. At the milestone date, analyse the work done and return to point 1. The analyse consist to sum the weight of each task done during the milestone. This sum represent the team velocity. This is useful for the next iteration, it allows to select a right number of tasks with a weight equals to this velocity.

2.4.3 ZenHub

ZenHub is a GitHub extension. It is integrated in the GitHub interface and add functionalities as a tasks board and a velocity management.

Create A Task

A strength of ZenHub is that it allows to transform an issue into a task. Figure 2.4 show the creation of an issue, it allows the following parameters:

- A title
- A description
- A pipeline. It is the column name of the dashboard where the task will be.
- A label. Example, a bug, a question,...
- A milestone.
- An estimate. It is a value representing the task complexity.
- Assignees. It assign one or more developer(s) to the task.

Tasks Board

Figure 2.5 shows the UCLCampus board. It contains the following column

- Student. All the tasks relative to the student section. This section only contains the tasks that are not done and that are not in the actual milestone.
- Campus. Same than student but for the tasks relative to the campus section
- Tools. Same than student but for the tasks relative to the tools section
- City. Same than student but for the tasks relative to the city section
- To Do. The task to finish for the milestone.
- In Progress. The task that are actually in progress.
- Done. The finished task during the current milestone.
- Closed. All the tasks done during the past milestone.

Milestone

The milestone creation take three parameters, A title, a start date and an end date. To put a task in the milestone, select the issue and set the milestone parameter on it. The milestone menu provide the list of tasks to do for the deadline. It provide an analyse about complexity point.

This repository

Search

amoyaux / UCLCampus

↩ Code

🔍 Issues 16

📄 Pull requests 0

📁 Boards

📊 Burndown

📖 Wiki

📈 Pulse

📊 Graphs

⚙ Settings

🔗 Pull requests

📌 Issues

📅 Gist

📌 To Do

🔍 Search (/)

New Issue

🏠 City 4

UCL Campus #3 Restaurants 5

UCL Campus #4 Bars 5

UCL Campus #5 Tourism 5

UCL Campus #6 Leisure activities 5

🛠 Tools 3

UCL Campus #7 Parkings 2

UCL Campus #9 Help for student 2

UCL Campus #10 Public transport 2

🎓 Student 0

📝 To Do 2

UCL Campus #13 Classes 5

UCL Campus #16 Implement webServices from OSIS 13

🔄 In Progress 3

UCL Campus #8 Maps 5

UCL Campus #21 Caledara 5

UCL Campus #20 Activy on campus 13

✅ Done 1

UCL Campus #25 Create maps for more campus : volume, mons, journal, etc enhancement

🔍 Search (/)

New Issue

| ID | Title | Status | Assignee | Labels |
|----|------------------------------------|-------------|----------|--------|
| 37 | Closed | 10 | | |
| 8 | UCL Campus #11 Lecture halls | Done | | |
| 8 | UCL Campus #18 Link to moodle | In Progress | | |
| 1 | UCL Campus #19 Link to udouvain be | To Do | | |
| 1 | UCL Campus #17 Student schedule | In Progress | | |
| 8 | UCL Campus #14 Login | To Do | | |
| 2 | UCL Campus #15 Implement database | To Do | | |
| 8 | UCL Campus #1 Languages | To Do | | |
| 8 | UCL Campus #22 Sports | To Do | | |
| 8 | UCL Campus #12 Libraries | To Do | | |

Figure 2.5: ¹⁵Tasks board

Chapter 3

Functionalities of UCLCampus

Now that the background has been set and that we have made our initial choices, the next step is to define what the application will actually do. The first section of this chapter describes our thought process when choosing the different functionalities of the application while the second section explains our reasoning when defining the user interface.

3.1 Choice of functionalities and sections

In order to define what kind of functionalities would 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.

“[...]

Quelques remarques:

- L'icône "Outils" du menu principal pourrait être renommé par soucis de clarté, on s'attend à tomber sur les "outils" ou paramètres de l'application et non pas du campus/ de la ville.
- Je ne vois pas trop l'intérêt de l'onglet mail étant donné que la plupart des étudiants possédant un smartphone ont déjà leur boîte mail étudiant connectée.
- L'onglet "Aide" est un peu caché et on ne s'attend pas à trouver ces infos là quand on clique, le lien parking devrait plutôt figurer dans le menu "ville" par exemple. (le renommer "infos utiles", ou qqch de plus parlant que "aide" éventuellement).

[...]”

This first comment mostly points out that the "Tools" section lacks clarity and that the functionalities we can find in this section are either unclear or useless. Another comment shared the same sentiment:

“[...] 2. la partie "mails" est sans intérêt puisqu'on est sur un smartphone où la messagerie est probablement déjà configurée.

3. le menu "Aide" pourrait alors remonter d'un niveau et remplacer "Outils" qui n'est d'ailleurs pas très parlant comme titre de menu”

This comment also points out that the "Tools" section should either be renamed or that it should contain other functionalities. Both commenters agree that the mail section does not have its place in the application.

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.

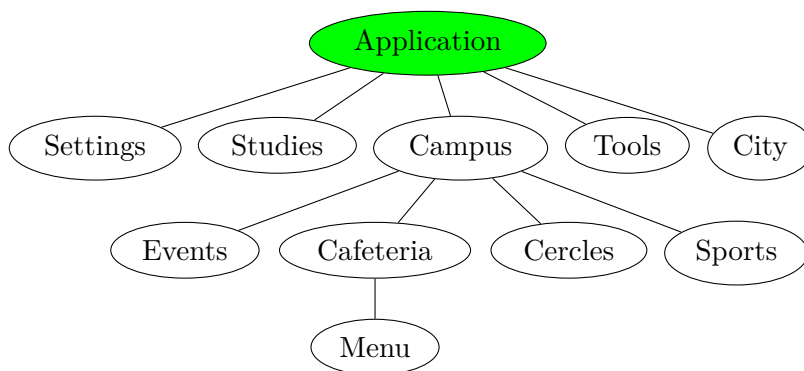
Chapter 4

Implementation

Once we had a clear idea of what the application was going to look like, we were ready to start implementing it. We started by designing a high-level view of the program which is detailed in the "Architecture" section. This section details

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.

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

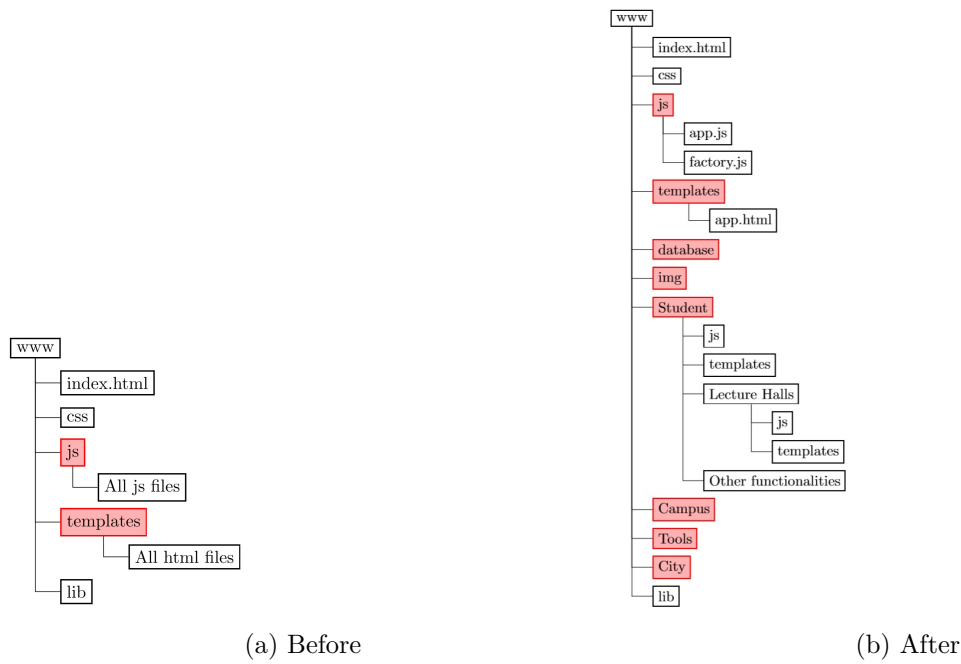


Figure 4.1: Folder evolution

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 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.

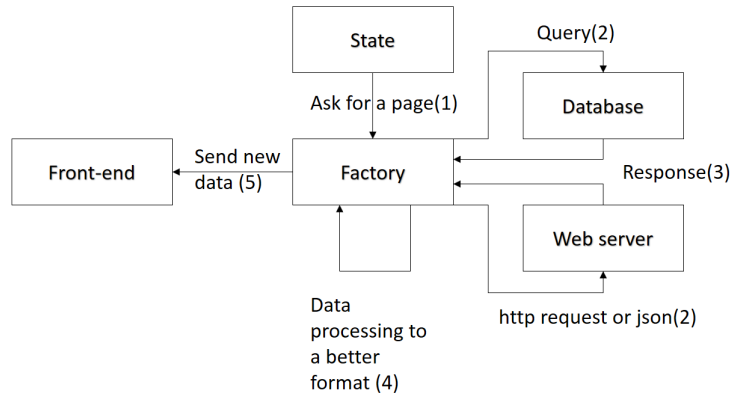


Figure 4.2: Factory operation

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.

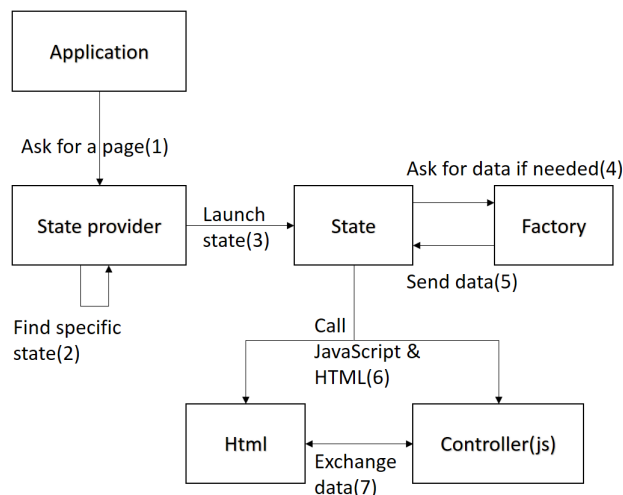


Figure 4.3: State provider operation

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 code¹. 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 |

4.3 Security

This section is not implemented yet because the web services from OSIS wasn't ready but if we had to implement it, that would be this way Three steps are needed for the security. First what do I need to protect? Second what are the threats and how the opponent can exploit them? Last, what do we need to implement in order to make it impossible?

Data to protect

The only data the hacker can be interest in, are the personal data of the user that means his password and username. He couldn't stole data of the database because the application his open source and they are so available on GitHub. An other thread would be the opponent to modify the data of the database but this is impossible because the database is replicated on local storage and the functionalities only communicate with that one (thus, the only database he can modify, is the local database on his smartphone). About our code, it's available on GitHub due to open source project and it's protected by a license. Therefore the only data that hacker could be interested to steal are the personal data of the user.

¹<https://www.uclouvain.be/459461.html>

Possible breaks

There exist two possible way to steal data. The first is if the opponent manage to get the hardware device, in this case, the opponent could try to access and steal data in the local storage. The second threat is to sniff the traffic between the application and OSIS, in this case he could intercept the package containing the request with the password and the username as parameters.

Protections

For the first thread, the password and the username should not be stored on the device (even if they are encrypted). So the solution to this problem would be to create the OSIS request directly when the user insert his credentials. With this procedure, we only store the student courses list and forget the credentials once the response is back. The courses list should not be accessible by an other person than the user but it is less critical than credentials, so in this case, it should be encrypted on the local device.

For the traffic sniffing, a solution would be to create a secure connexion with OSIS where packets are encrypted. For example an RSA encryption with public and secret key, UCLCampus would send a first request to obtain a public from OSIS, encrypt the request with it, send it back to OSIS which can decrypt it with the secret key. Due to this procedure, the opponent can catch the packets between OSIS and UCLCampus but can't read or do anything with them (except create false request and brute force password and username but this can be done on the website uclouvain.be).

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)

OSIS web services are not implemented but a dummy server has been implemented to simulate its behaviour. The SGSI give us the JSON response format.

```
Student[{
  registration_id
  FirstName
  LastName
  Offers:[{
    Acronym
    Title
    Learning_unit:[{
      Acronym
      Title
      Credits},
    {...},...]
  },{..},...]
}]
```

This represent a student with a first-name, last-name and a registration id. The offers represent a learning option. Example, BAC1 in computer sciences. A learning unit represent a course.

We create a query with the student user-name and password as parameters to obtain this response. The response can be automatically transform from JSON to a JavaScript object. The JavaScript is stored and used in other functionalities.

If the username and password in the query don't match any student, it return a empty response.

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 is 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.louvaininfo.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://louvaininfo.be/evenements/feed/calendar/'
```

The result is then automatically parsed and stored in a JavaScript objects 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. The list of course acronym used come from the OSIS response. It is the list of acronym for all leaning unit in the actual offer of the student. .
- `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 website². 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) and 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>

³http://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 the first time we will explain the design. Secondly, we will explain how it works from a technical point of view. All the page screen described in this section can be found in the annexe X. The application has a size of 50 Mo.

5.1 The application UCLCampus

5.1.1 Header

design We create a global header for the application. The header has different colours depending on the section (Chap 4.2.3 for more details). Otherwise, it has three components, the first is the name of the page. For example "Sainte-Barbe" if we are looking at the details of this one. The two other components are buttons, the first is the back button, it allows to back to the previous page. The second button open the settings menu. This header is by default enable on all the pages of the application but the programmer can overwrite it if he wants to add something, such as new buttons, search bar,...

Implementation Ionic framework provides a file named 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 pieces of a user interface that will be available everywhere in the application. These pieces can be overwritten. 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 a single code for every header except if they have been overwritten. Thus it can be modified quickly and efficiently. The implementation of the header will not be described here given its lack of interest. It's only html and css code with components provided by Ionic.

5.1.2 Footer

Design The footer is a bar with buttons below the screen. As the header, the footer colour change depending on the section. There are four buttons on the footer, the four main sections (student, campus, tools, city). We thought it was more user-friendly to do that because you don't need to click on the previous page until you reach the main menu if you want to explore another section. For example, if you are looking at the details of a lecture hall and you want to see the events of the week, without footer you need to click three times on the back button to reach the home menu 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 points out in which section you are, it will greyed out all the buttons except the one representing that one.

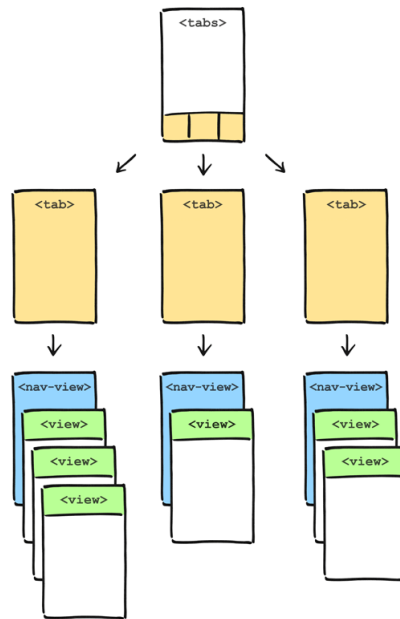


Figure 5.1: Tabs diagram

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 with 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 example, if we are on the libraries' page, there exist a state associate to it, informing that this page belongs to the student section. From that, the framework manages automatically to highlight the student button in the footer in order to show the user that he is in the student section. For more details¹ see figure 5.1, it creates a tree structure of sections, or tabs in the schema. Each section holds a list of views (pages) and his own navigation history stack. The history stack matter for the back button function.

5.1.3 The Homepage

When the user launch UCLCampus on his smartphone, he arrives 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 opens the application. Another choice could be to put him in the student section and let him use the footer button to navigate between sub menus, we found this solution less comfortable for the user. 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. No technical details will be given for this page due to lack of interesting parts. It is created with classic HTML, JavaScript and css code.

5.1.4 The Student/Campus/Tools/City Menus

This four sub menus have the same design and are implemented the same way. TODO lien vers l'annexe des screens ou alors importer l'image.

Design The menu offers a list of buttons. Each button is composed of its functionality name and a representative icon next to it. We choose to remove the back button of the header because

¹<http://ionicframework.com/docs/api/directive/ionNavView/>

it was useless to go back to the main menu. Instead, the user can use the footer buttons to access a new section.

Implementation The four menus have the same code skeleton. The principle is simple, each menu has his own factory. This factory holds a list of objects, each object represents a button. These buttons 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 parameter 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 on the menu, if not the button is not 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 passed 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 license²), 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.

²<https://market.ionic.io/plugins/onezone-datepicker>

³<https://github.com/EddyVerbruggen/Calendar-PhoneGap-Plugin>

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.

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.

⁴<http://cordova.apache.org/docs/en/3.0.0/cordova/inappbrowser/inappbrowser.html>

Events

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

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 louvaininfo.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.

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.

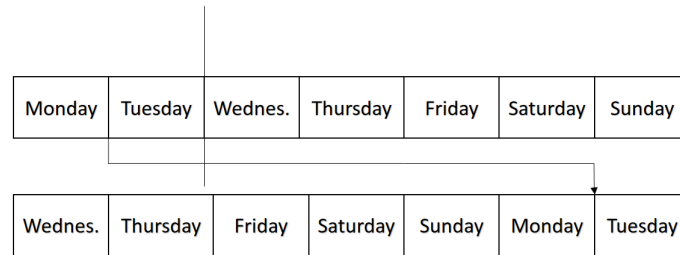


Figure 5.2: Sorting day operation

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 llm maps ⁵ . We created the menu in order to introduce new functionalities later.

LLN Maps

Design llm maps has its own header and footer, after a discussion 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

⁵<https://github.com/mathieuzen/llnmaps>

- 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.

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"
});
```

⁶<https://github.com/angular-translate/angular-translate>

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 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 use 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:

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

Finally, create a symbolic link for node:

```
1 $ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

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

```
1 $ node -v
2 v0.10.25
3 $ npm -v
4 1.3.10
```

Step 3: Installing Ionic and its dependencies

If you are on Windows, drop sudo from the following commands.

First, install Cordova:

```
1 $ sudo npm install -g cordova
```

Then, install Ionic:

```
1 $ 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: Forking the project

First, navigate to <https://github.com/amoyaux/UCLCampus>. Click the Fork button in the top right corner. Note that in order to fork the project, you will need a GitHub account. Once you have forked the project, open a terminal into the repository of your choice. Then, enter the following command:

```
1 $ git clone https://github.com/<your username>/UCLCampus.git
```

This will clone the project files into the chosen repository.

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:

```
1 $ cd UCLCampus/UCLCampus/
```

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

```

1  $ npm install
2  $ cordova plugin add cordova-plugin-device
3  $ cordova plugin add cordova-plugin-console
4  $ cordova plugin add cordova-plugin-whitelist
5  $ cordova plugin add org.apache.cordova.network-information
6  $ cordova plugin add https://github.com/brodysoft/Cordova-SQLitePlugin.git
7  $ cordova plugin add https://github.com/litehelpers/Cordova-sqlite-storage.
   git
8  $ cordova plugin add https://github.com/EddyVerbruggen/Calendar-PhoneGap-
   Plugin.git
9  $ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-
   inappbrowser.git
10 $ cordova plugin add cordova-plugin-geolocation
11 $ cordova plugin add https://github.com/brodysoft/Cordova-SQLitePlugin.git
12 $ cordova plugin add https://github.com/an-rahulpandey/cordova-plugin-dbcopy.
   git
13 $ 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:

```

1  $ 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:

```

1  $ ionic platform add android

```

or

```

1  $ 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:

```

1  $ ionic run android

```

or

```

1  $ 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:

```
1 angular.module('ionicApp').controller("MyFunctionalityController", function($scope) {  
2     $scope.textToDisplay = "This is the text we want to display."  
3 });
```

Step 3: Writing the HTML code

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

```
1 <ion-view title="Title">  
2     <ion-content>  
3         <div>  
4             {textToDisplay}  
5         </div>  
6     </ion-content>  
7 </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:

```
1 .state('app.tools', {  
2     url: "/tools",  
3     cache : false,  
4     views: {  
5         'tools-tab' :{  
6             templateUrl: "tools/templates/tools.html",
```

```

7     controller: "ToolsController"
8   }
9 }
10 })

```

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

```

1 .state('app.myfunctionality', {
2   url: "/myfunctionality",
3   views: {
4     'tools-tab' :{
5       templateUrl: "tools/my\_functionality/templates/my\_functionality.html",
6       controller: "MyFunctionalityController"
7     }
8   }
9 })

```

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

```

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

```

Below these lines, add the following:

```

1 <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:

```

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

```

Change it to the following:

```

1 toolsMenuList : [
2   { title: 'Maps' , icon:'icon ion-map royal', url:'app.maps', campus:['
3     Louvain-la-Neuve' ]},
4   { title: 'My functionality' , icon:'icon ion-help royal', url:'app.
5     myfunctionality', campus:[] }
6 ] ,

```

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.

5.2.6 Step 5: submitting your functionality

Once you have completed a functionality and believe it is ready to be integrated into the application, you can send a pull request to the project owner. The pull request will then be reviewed and will be added to the application if it is accepted. For more information about forks and pull requests, please refer to: <https://help.github.com/articles/using-pull-requests/> and <https://help.github.com/articles/fork-a-repo/>.

5.3 Future functionalities and possible improvements

In this section, we will review the functionalities we couldn't add to the current application for various reasons, as well as looking at ways we could improve the existing functionalities.

5.3.1 Future functionalities

The "City" module

In chapter 3, we described the "City" module where a user could find information about museums, cinemas, restaurants and so on. This module was not implemented yet for different reasons. The challenging part about such a module is that some restaurants might not want to be listed in the application. Another challenge is the lack of any database concerning the restaurants of a given campus. It would therefore be necessary to list all restaurants, museums and other activity centers of all campuses and possibly ask them whether they want to appear in the application or not. While this module would certainly be interesting for the user, it would also be hard to keep the information up to date without having someone responsible to update the application every year or so.

"Cercles" and "Kot à Projets"

The "Cercles" and "Kot à projets" functionalities were also supposed to be implemented but weren't for reasons similar to the city module. It would also be necessary to list them and ask them if they want to be listed in the application, as well as maintaining the database up to date since "Kot à Projets" change almost every year.

Integrating OSIS

At the time of writing the UCL was changing its infrastructure. Integrating OSIS, the new architecture, into the application was scheduled but sadly couldn't be achieved. OSIS was going to be used to fetch the courses a student is enrolled in at a given time and displaying information about said courses. For now the information is given by a dummy server and doesn't correspond to the reality. An interesting thing to do would therefore be to integrate OSIS to the application as soon as possible. It is indeed our belief that this functionality is of paramount importance if we want the application to be successful.

Tools

The "Tools" module currently only contains the "Maps" functionality. Other functionalities could be added to this module, such as a functionality regarding how to reach a campus using common transportation, or how to park in the city.

5.3.2 Possible improvements of existing functionalities

Adding support for all campuses

A lot of functionalities are only supported by a few campuses so far. We list here the functionalities that would benefit from being supported by multiple campuses:

- Lecture halls
- Libraries
- Events

- Sports
- University restaurants
- Cercles
- Kot à projets
- The entire "City" module
- Maps

5.3.3 Other improvements

- This project currently uses the 1.3 version of Ionic. It is however already possible to use Ionic 2 and migrating the application to this new version could be interesting. At the time of writing Ionic 2 was only available as a beta release, which is why we didn't use it immediately. Ionic 2.0 is interesting because it deploy tools for the tree architecture that is used (see chapter 4.1).
- An other improvement is to create a logo and a splash screen for the application.
- Create the Dutch translation. Everything is implemented, it is missing the map between variable and their translation.

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

The UCLCampus implementation brought out our Ionic benefits and drawbacks.

6.1.1 Benefits

- Easy to learn and use. The tutorial on their website is complete and bring possible architecture notions. Combined with a tutorial about angularJs ¹, it is possible to start an application.
- The css and JavaScript code provided are useful and well designed. Around 90% of the css components and icons used come from it.
- A large community that provide quick response and many example for simple topics. Moreover, this community provide a great plugins panel, it was never necessary to create our own.
- Well documented. Every component possible attributes are listed and described and some basic example are provided.
- AngularJs provide a rich architecture to modulate the source code.
- Easy to handle web services and database management.
- Correct performance for a container approach. See section 6.3.1 for details.

6.1.2 Drawbacks

- No response for advanced topics.
- A poor architecture for HTTP requests to external website. Ionic removed "cookie" and "set-cookie" in the response parameters. That is a real problem when a website use a cookie authentication or parameters inside the cookie. The reason is that they implemented a local cookie storage which is more efficient. However, it only work for request inside the application.

¹We used w3school tutorial <http://www.w3schools.com/angular/>

- Still in development. Some plugins and components still in development. They contain bugs and should not be deployed for an end user.

6.2 Testing

6.2.1 Unit Test

6.2.2 Performance testing

In order to assess the performance of our application, we profiled it using Google Chrome. Google Chrome indeed has a tool, available using the `chrome://inspect` address, that allows developers to profile an application running on their Android device. The following results are therefore only relevant for Android phones. The device we used for these tests was a Huawei P8 Lite, a smartphone with average performances (source benchmark). Results may vary with higher or lower end smartphones. Despite the limitations of our testing, the results are still interesting and we will present them in this section.

Profiling

We used Google Chrome tool in order to profile several functionalities of our application. It is important to note that monitoring and recording the application's activity adds a bit of overhead and the results we show here will be slightly worse than the ones one would obtain without any monitoring tool. However, the overhead is not important enough to invalidate the results. The idea is to start the profiling, click a button and wait until the next view is displayed. This gives us an idea of how long a user has to wait between views. Figure 6.1 shows the result we obtained when clicking on the "Libraries" button in the Student Menu. The rest of the profiles are available in annex X (TODO).

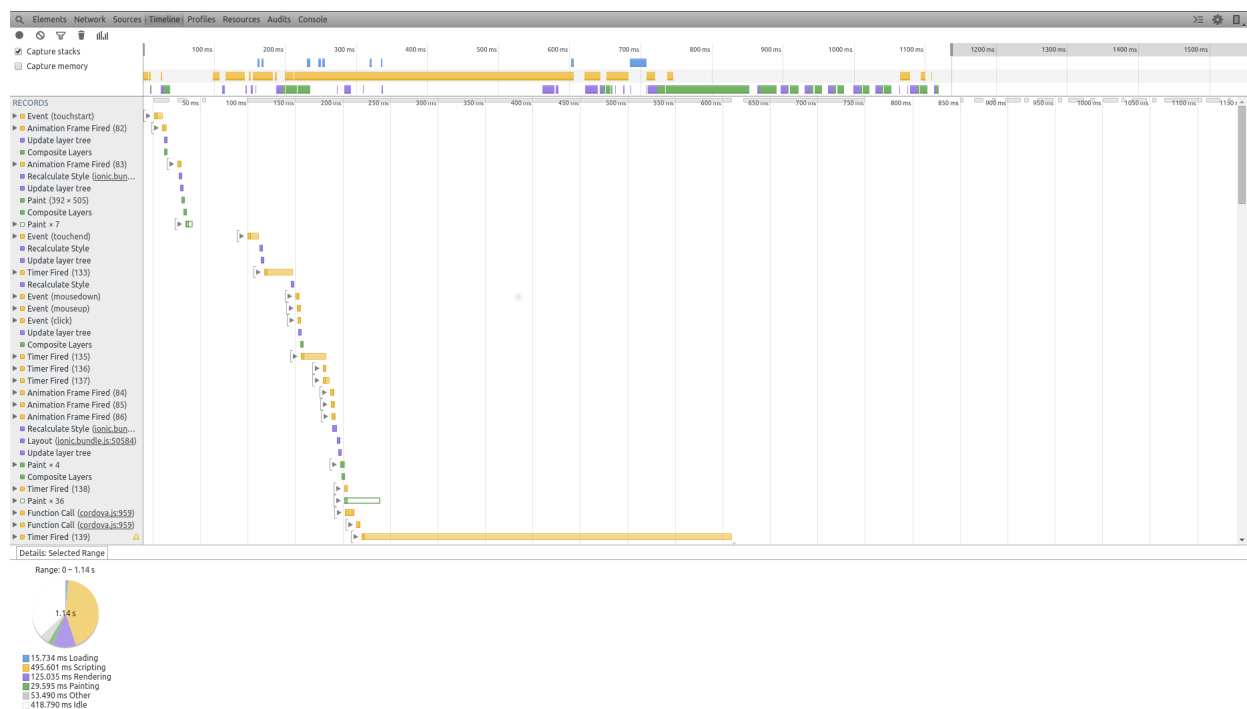


Figure 6.1: Profile of the Libraries functionality

In this figure we can see several interesting things. We start by observing that the application

takes approximately 1.1 seconds to display the list of libraries. We ignore the last 0.04 seconds displayed in the figure as the work was done and the rest of the time was spent idle. The rest is split among several tasks:

- Scripting : Running the JavaScript code. Time spent: 495.601ms, 45% of total time
- Loading : Loading different resources (images, ...) from memory. Time spent: 15.734ms, 1.4% of total time
- Rendering : Organizing the page layout. Time spent : 125.035ms, 11.4% of total time
- Painting : Displaying the view. Time spent : 29.595ms , 2.7% of total time
- Other. Time spent : 53.49ms, 4.9% of total time
- Idle. Time spent : 385ms, 35% of total time

We can see that, despite displaying several images, the application only spends 14.1% of its time to render and paint said images. Instead, most of the time is spent scripting. This means that the most time consuming thing the application does when displaying the libraries is finding said libraries in the database and converting the result to a JavaScript list. This behavior seems to remain the same for every functionality of the application. Indeed, scripting is usually the main task in every profile, ranging anywhere from 30 to 60% of the total execution time. We will now look at the execution times of other functionalities: //TODO make a table ??

- Changing tabs : 489ms
- Displaying the schedule : 2005ms
- Displaying the list of sports : 4260ms
- Displaying the student menu : 700ms
- Displaying the cafeterias : 1051ms
- Displaying the campus menu : 970ms
- Displaying the events : 2890ms
- Displaying the details of an item (library, lecture hall, ...): 700ms
- Displaying the list of lecture halls: 1290ms
- Displaying the map: 3650ms

From these results, we can observe 3 types of functionalities:

- Simple display of information already available. Takes less than a second. Examples: changing tabs, loading a menu, displaying details of an item.
- Display of information available in the local database. Takes between one and two seconds.
- Display of information available online and maps. Takes more than two seconds.

These results make a lot of sense since retrieving information online means we have to wait for the server to respond, which explains the overhead. There is not much we can do to avoid waiting for the server aside from caching the results in the application. Caching the results is also possible for functionalities retrieving information from the database. This technique significantly increases performances but only does so after the user already used the functionality a first time. We observe the following performances once the results have been cached:

- Displaying the list of lecture halls: 834ms
- Displaying the list of libraries: 782ms
- Displaying the cafeterias: 754ms
- Displaying the schedule: 1530ms
- Displaying the list of sports: 1350ms

We can observe a significant performance increase once the results have been cached in these functionalities, usually around 500ms and almost 3 seconds for the list of sports. This is due to the slow response time of the server we contact to retrieve the sports. The other slower functionalities (maps and events) cannot be easily cached. Indeed, it does not make sense for maps as there are no results to be cached. As for the events, the list is constantly updated and caching the result might lead to inaccurate information being displayed. We can now see that, once the relevant information has been cached, most functionalities respond in less than a second. We could not find any standard regarding the recommended response time for an application. However, Android will consider the application as not responding if it stuck for more than 5 seconds. As we can see, none of our functionalities exceed this limit, even when not cached. According to [source TODO], a response time ranging between 500 and 1000 ms is considered to be immediate, which is the recommended interval to load pages. Most of our functionalities are in this range. In order to make the wait for slower functionalities more bearable, we added an indicator that the functionality is loading. That way the user still receives feedback to indicate that his action, in this case the press of a button, has been registered and that the application is working on it. We conclude that, despite using a hybrid framework, the performances of our application are acceptable and that most users will not notice any delay when using the majority of the functionalities. Furthermore, the slowest functionalities often depend on the response times of external servers.. In those cases, using a native approach would not have helped performance-wise.

Chapter 7

Conclusion

Chapter 8

Bibliography

<http://www.gnu.org/philosophy/open-source-misses-the-point.html>
https://en.wikipedia.org/wiki/GNU_General_Public_License

