ITI Bachelor Project

August 2018

PocketHepia
A student card platform based on NFC

Maxime Alexandre Lovino

Supervised by
Prof. Mickaël Hoerdt

h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so GENÈVE

Haute Ecole Spécialisée
de Suisse occidentale

Contents

List of Figures

Acknowledgements

Terms and Definitions

IDE est un acronyme pour Integrated Development Environment. Il s'agit d'un programme intégrant un éditeur de texte avec coloration syntaxique et autocomplétion du code pour un ou plusieurs langages de programmation ainsi que des fonctions de compilation, débogage et tout autre fonctionnalité permettant de faciliter et fluidifier le travail du développeur.

JVM est un acronyme pour Java Virtual Machine. Il s'agit d'une machine virtuelle permettant d'executer un programme compilé en bytecode Java sur un ordinateur ou un terminal mobile. On parle de langage fonctionnant sur la JVM lorsque la compilation du langage produit du bytecode Java..

NFC est un acronyme pour Near Field Communication qui représente un ensemble de protocoles de communication permettant de communiquer à des distances de quelques centimètres en utilisant l'induction électromagnétique.

Chapter 1

Introduction

Since starting my studies at HEPIA almost three years ago after two years at EPFL, I've been shocked by the lack of commodities and study spaces for students. Some of this lack is due to the difference in scale between the two schools, one of them being composed of mainly three building in a constrained city environment and the other an almost-autonomous campus outside the city. But, actually, there isn't really a lack of space at HEPIA, but a lack of space that students can use to study. Most of the classrooms are locked when not in use by a teacher. When asking about why that is the case, I've been told that it was mainly for security concerns because of the equipment present inside the rooms. If the rooms stayed unlocked, there was no way of knowing who stole or broke something. I suggested the idea of giving access to select students to these classrooms to study and work on projects but it wasn't practical because copies of keys had to be made, the students had to make a money deposit to make sure that they didn't run away with the keys etc.

The real solution would have been to use our student cards as an electronic door key to access the rooms, as well as enable other uses for the cards, such as using them as electronic wallets to simplify the payments at the canteen during lunch break. When I suggested the idea, people mostly laughed at me and said "[...] they're never going to do it, unless someone actually does it, presents it as a finished product and then they decide to use it." So, here I am, after 3 years studying at HEPIA and for my Bachelor Project I decided to work on this exact idea. A multi-function student card platform built on NFC technology with accompanying mobile and web application for administrators to manage the platform and for students to monitor their usage statistics.

1.1 Project inspiration - Camipro EPFL

1.1.1 The card and the official core platform

The inspiration for this project mainly comes from my experience studying at EPFL. The EPFL student card, named Camipro[2], contains an RFID chip that allows to perform

various tasks that simplify student life on campus. The use cases for the Camipro card are the following:

- Electronic wallet to pay at every canteen on campus as well as select third-party retailers (for example the Migros shop present on campus)

- Access key to unlock doors and buildings

- Card to collect documents sent to the centralised printing pool system at any printer on campus

- Rent public bicycles[1] on campus and in the Lausanne area

- Rent cars by linking a Mobility[2] subscription to the card

- Borrow books at the library

- Use electric car chargers on campus

- Turn on booked electrical barbecues present on campus[3]

An accompanying web platform called MyCamipro was built as part of the Camipro launch to manage and activate the different services on the card as well as see the recent transactions and the rooms we were given access to.
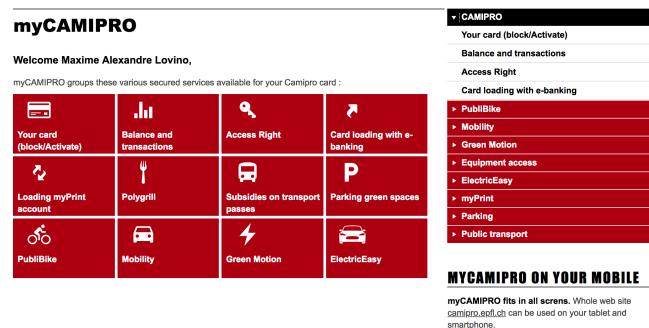
Figure 1: Screenshot of the MyCamipro website

1.1.2 PocketCampus

With the increased usage of smartphones by students, in 2010 a team of 20 computer science students decided to build the PocketCampus application as part of a software

---

[1]All students have access to a free Publibike account on their card `https://www.publibike.ch/en/publibike/`

[2]`https://www.mobility.ch/en/`

[3]The service PolyGrill allows students to book free barbecues on campus and access them with their camipro `https://camipro.epfl.ch/cms/site/camipro/lang/en/polygrill_electric_barbecues`

engineering class. They continued working on the project after the academic project was finished and it became the official EPFL application in 2013. [3].

Initially you could mainly see the balance of your Camipro card on the app, but version after version, the development team added new integration in the app by collaborating with different services at EPFL. These functions include:

- Accessing the menus of all canteens on campus

- Searching through the whole EPFL directory

- Having access to IS-Academia data to see course schedule and grades

- Printing from your smartphone on the EPFL print system

- Accessing Lausanne public transportation itineraries

- Accessing Moodle documents

After having integrated every requested features, the team launched a beta web version of PocketCampus for EPFL in June 2018. They also started diversifying their business by working on PocketCampus as a platform that can be integrated in other companies and stopped working exclusively with EPFL. They announced plans on partnering with Lausanne University (UNIL) to integrate their platform there.

Figure 2: Screenshot of the web version of PocketCampus

Figure 3: PocketCampus Android application
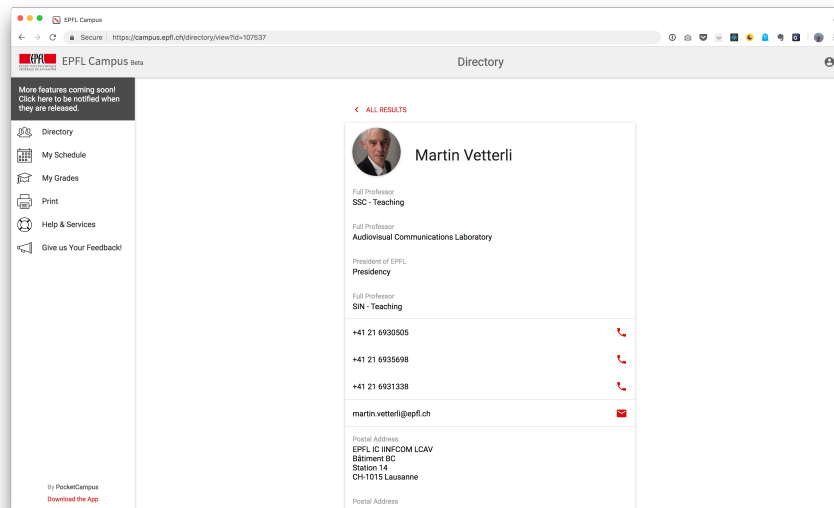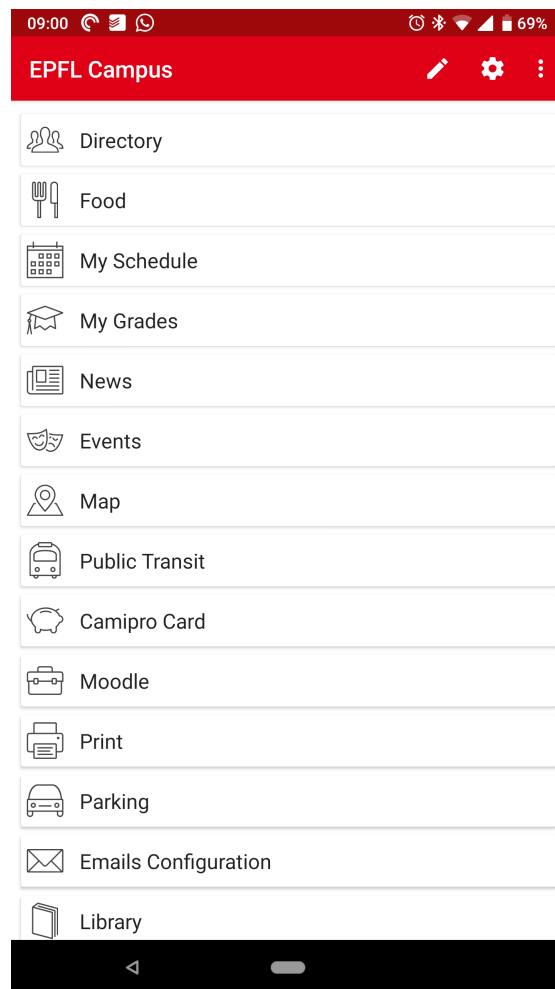
Chapter 2

The project

The project is called PocketHepia, reminiscent of the name of the project at EPFL and will consist of the two parts of the EPFL project presented earlier: the student card platform, as well as the applicative platform to access information.

The idea of this project is not to build as many features as the EPFL platform due to the time constraints inherent to the Bachelor Project, but rather to focus on some key aspects of the card at first, namely payments, access control and library. We are also building features on the payment side that are not available on the EPFL platform, for example the ability to send money between users. The project of course will be extensible with other features outside the scope of the Bachelor Project.

In this chapter, we will present the main features, roles and user stories for the project as specified at the beginning of this one. All of these are part of the project, but not necessarily part of the Bachelor Project. The idea was to specify the entire project and to start implementing a subset of the functionality for the academic project and continue working on the rest as well as new features outside this scope. Mainly we wanted to remain independent of other school services and infrastructure because of the associated time it would take to get permissions and setup integrations, as well as restrictions to development environments to secure access to these services. For each element of this chapter, we will specify whether it is going to be implemented as part of the Bachelor Project.

The project is composed of three main components:

- The physical student card

- An administration component

- An user facing component

A mobile application and a web application will be developed and both will offer the same user facing features as well as specific administrative features relevant to each application.

2.1 Features of the app

As stated before, we decided to limit the project to the three most important features in our opinion and, out of these, two are going to be implemented as part of the bachelor project.

2.1.1 Payments

The first feature is the electronic wallet functionality. We decided to start with this one because it would be bring the most benefits to the students and would not need specific hardware or collaboration with any school service. Every user has an electronic wallet linked to its account on the platform and can then send money to any other user by choosing the amount to send and tapping the user's card on his phone. Users can be assigned a specific role (see section 2.2) to allow them to receive money directly from a card.

There will be two main ways to add money to an user wallet. Either the user can give cash money to a person with administrative role that will then add money to the user's balance. Or the user will be able to add money to its account by using its credit card on the web application.[1]

2.1.2 Accesses

To continue, the second element of the project is the handling of access control through the platform. This was the original pain point we noticed at HEPIA with the lack of an easy way to handle accesses to the available classrooms. We will be able to create different areas to group rooms inside them. For example, an area could be an entire floor or labs for a specific section of HEPIA.

As far as giving access to users, we can give access to a room from a given date and specify if needed an end date for the access. For example, we can setup access for a user until the end of the current semester. We also decided to introduce an optional time range during the day during which the given access is active, so we can for example allow a student to enter a room only between 8am and 7pm.

We also specified the ability to delegate the administration of an area to a user, so that access to rooms can be handled at a department or section level in the school for example (see section 2.2). This feature is planned but will not be implemented as part of the Bachelor Project.

---

[1] At the moment, the project is still in the development phase so no bank accounts have been linked to this project and you will be able to recharge your account only using a specific test credit card generated for development purposes

2.1.3 Library books

Finally, the last feature of the project would be an integration with the library to be able to use the card as a library card. Our approach at first was to think about creating the book loans on our platform with the ISBN to identify the book and integrate with the REST API built for BibApp but it wouldn't have been very effective because the library already handles the loans through the NEBIS platform.

The correct way to handle this would be to link the library NEBIS identifier for every user with their account on our platform and then ask NEBIS for API Access to their platform to retrieve book loans for all users and display them on the mobile and web application. This would also benefit people working at the library as they would not need to change their current workflow to accommodate.

We don't have the time during the bachelor project to start a discussion with NEBIS to get access to the required information so this feature will have to be implemented later on.

2.2 Roles

We defined a set of roles for the users on the platform. First, every member of the platform is a simple user. This means that the person has an account on the platform, can login to the web and android application and has read access to its payments, its accesses and its other information and can send money to other users.

Then, there is the admin role that can be added to an user. This role enables the creation of users, attribution of roles, the consultation of administrative logs as well as the creation of access components (rooms, readers) and the attribution of accesses to users.

While these two roles would have been enough to handle all our features. We decided to specify roles specific to different components of the platform.

One of these roles is the "Accept payments" role. This is specific to a canteen or a shop that wants to handle payments using the student card. While every user can send money to another user from the mobile app by tapping the other user's card, this isn't very practical for a canteen or a shop where the transactions should go the other way around. So the shop creates the payment and taps the user's card to take money from it. At first, we wanted this feature to be available to everyone but we thought about security concerns with this solution because a student could create a payment from the app and start tapping his phone on lost cards or even on people's pocket and if the card was detected it would "steal" their money. So by enabling this feature as a role, we could only allow trusted people, such as canteen's owners to receive payments in that way.

Furthermore, due to the introduction of GDPR recently, we created a specific role called "Auditor" to access sensitive logs concerning all users, mainly access logs and transaction history. Only a user with this role can view all transactions between users and access logs for every room.

Then, there are two roles that will not be implemented as part of the Bachelor project, the "Can invite" and "Area admin" role. The first is to allow specific users to create temporary accounts without needing to contact an administrator. A use case for this would be a teacher creating a temporary account for a visiting colleague from another school. The "Area admin" role consists of delegating the administration of the accesses for an area to an user. Similar to the concept of DNS zones delegation, an administrator could for example give this role to the ITI section dean to allow him to give access to the rooms present on his floor.

Finally, the last role is the "Librarian" role that as its name implies is attributed to users working for the library. This role enables the creation of books loans at the library for users. As stated in section 2.1.3, this role will certainly be removed completely because there is already a system in place to handle book loans at the library.

2.3 The models

2.3.1 User Accounts
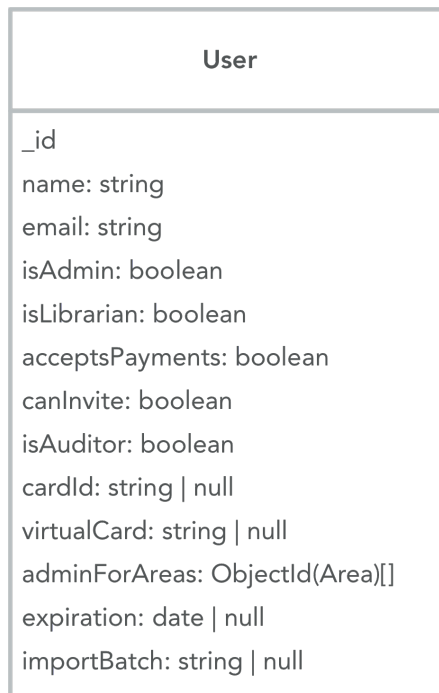
Figure 4: User Model

The first of our models is the User. A user collection is created to store all users on the platform. Users are identified by their email address and will login with their email and password, their full name is also stored. The authentication specific fields will be added to the model by Passport.JS (see section 2.5). A boolean field is also stored for each permission.

The expiration date, the virtual card and the delegation of admin zones are not implemented as part of this bachelor project.

2.3.1.1 Integration with existing AAI user accounts

If this project is put in production at HEPIA, we will have to only change this model to integrate it with an existing authentication service, such as AAI or with a directory LDAP service.

In that case, we would remove information already present in the authentication service such as name and email and just store the unique identifier from the authentication

service in our model. This model would act as an augmented database on top of the existing service. We would then forward all login/password authentication request to the authentication service and would create the entry for the user in our database after the first login.

This is inspired by the way Nextcloud[2] for example stores user accounts that are linked to a directory service, for example an Active Directory. It stores the full Distinguished Name from the directory service as the unique identifier for the account to link the local account to the directory account.

While we could have asked Switch-AAI to integrate AAI authentication in our application, the time constraints of the project were too tight to launch a discussion and we wanted to stay independent of other services in this phase to avoid being slowed down by for example needing to run our servers on the HEPIA network exclusively to access the directory.

2.3.2 Payments

Transaction
_id
title: string
date: date
from_user: ObjectID(user) | null
to_user: ObjectID(user)
amount: number
stripe: boolean
adminCharge: boolean

Figure 5: Transaction Model

To handle payments, we defined a Transaction model that represents a money transaction. We store the id of the `to` and `from` users instead of embedding the users document inside the transaction in order to avoid using too much space in the database. The number of transactions can grow very high and it is a waste of space to store two full user documents in each transaction[3]. Finally, we handle the two type of recharges for the user account by setting the `from` user to `null` because the money does not come from any user account and set the corresponding `stripe` or `adminCharge` field to true.

---

[2]`https://nextcloud.com/`
[3]For more on that, see section 5.2 on Mongo References

The balance of a user is calculated from the list of transactions in which the user takes part. It is calculated as the sum of the transactions in which the user is the beneficiary minus the sum of transactions in which he makes the payment.

2.3.3 Accesses

Area
_id
name: string

Room
_id
name: string
accessReaders: string[]
area: ObjectID(area)

Access
_id
user: ObjectID(user)
room: ObjectID(room)
startDate: date
endDate: date | null
startTime: number
endTime: number
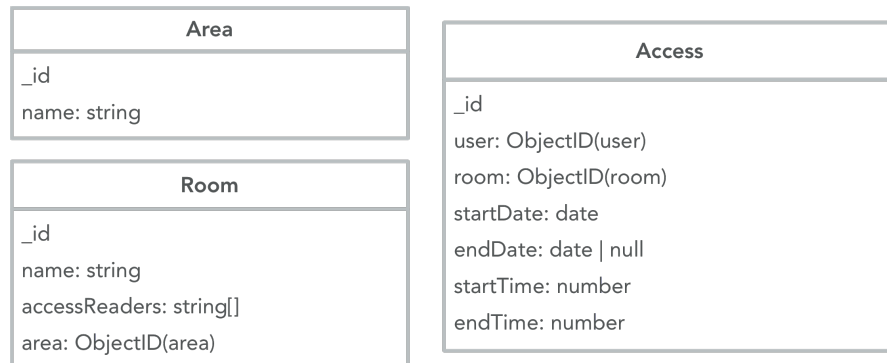
Figure 6: The three models handling accesses

2.3.4 Logs

2.4 User stories

From the beginning, we wrote simple user stories as a roadmap for the project features we wanted to implement. All of these features are not part of this academic project but are part of the broader student card project.

2.5 Technological choices

2.5.1 The data and backend

2.5.1.1 Realm vs. Express-MongoDB

2.5.1.1.1 Advantages of Realm

2.5.1.1.2 Specific use cases

2.5.2 Mobile application

2.5.2.1 Why the need for an application?

2.5.2.2 Why go native?

2.5.3 Web Frontend

Chapter 3

NFC and NDEF

3.1 Introduction to NFC

3.2 NDEF Format

3.2.1 Text payload

Chapter 4

Developing for Android

Android is a mobile operating system developed by Google and launched in September 2008. The core of Android is open source and is known as the *Android Open Source Program (AOSP)*. But nobody really knows or uses the AOSP version of Android, every phone manufacturer, even Google, forks the AOSP version to build its own for their devices. What is often referred as "pure Android" is the Google implementation of Android on their Nexus and Pixel lines of smartphones. Android runs on all kinds of platforms, from mobile phone to watches, as well as tablets and laptops.

A SDK and an IDE are provided for developing Android applications. Those applications are written mainly in Java, with the ability to write native code in C++ using the Android NDK[1] and interact with the Java code using JNI. The support for a second language named Kotlin was announced at Google I/O 2017 (see section 4.2). Applications run in a virtual environment on Android called ART. You can think of ART as the equivalent of the JVM for desktop Java development.

4.1 Android Fragmentation

One of the biggest problems for developers is the fragmentation of OS versions running on Android devices. As of May 2018, only 62.3% of devices were running Android Marshmallow (version 6) or later. Android Marshmallow was released in 2015, meaning that 37.7% of devices were running software that was more than 3 years old without security updates or new features (see table 4.1).

---

[1]This is used for example to use some C++ libraries, like OpenCV.

Year	Version Name	Usage of this version	Usage of this version or later
2010	Gingerbread	0.3%	100.0%
2011	Ice Cream Sandwich	0.4%	99.7%
2012	Jelly Bean	4.3%	99.3%
2013	Kit Kat	10.3%	95.0%
2014	Lollipop	22.4%	84.7%
2015	Marshmallow	25.5%	62.3%
2016	Nougat	31.1%	36.8%
2017	Oreo	5.7%	5.7%

Table 4.1: Android OS Fragmentation (May 2018)[4]

4.1.1 The problem with updates

This problem is inherent to the very nature of the Android operating system. Each phone manufacturer, or OEM, can fork its own version of Android and integrate its own skin and set of apps on top of it. When a new version of the operating system is released by Google, they can't just start using it. They have to first adapt and test all their apps and customisation with the new version before it can ship to the devices. This is a time (and by extension money) consuming process and many OEMs just don't care about maintaining their devices for more than one or two years. To make matters worse, in certain countries, such as the United States, mobile carriers have to validate and apply their own custom apps and settings on top of the OS, adding to the time needed to validate an update and causing a supplementary potential roadblock to the release of an update.

4.1.2 What Google is doing about this?

Google has in the recent years taken different actions to ensure that most of the devices run safe and up-to-date software on them without depending on the OEMs willingness to maintain their products.

4.1.2.1 Updating core apps through the Play Store

One of the most successful changes to Android in recent years has been to slowly move all Android core applications that are present on every Android device[2] to the Google Play

---

[2]Actually, not every Android device. Some Asian markets, mainly China don't have access to these apps because they don't have access to any Google services. This is an edge case that won't be discussed in this document.

Store. These apps include for example Gmail, Google Calendar, the browser (AOSP browser and Google Chrome) and even apps like the Phone Dialer and Contacts apps. This move to the Play Store allows Google to update these applications more frequently without needing a full operating system update. While it may be seen as a necessary evil at first, because it is the only way for them to update these apps if the OEM don't apply operating system updates, it is actually a very useful move because it allows for faster iteration on these applications and quicker response for bug fixes.

If we compare this to the other major mobile operating system, iOS, all main applications on the platform are bundled with the OS. So, if Apple needs to update the Safari browser to support a new web API they have to release a full OS version and push it to all their devices instead of just updating the application that needs an update as Google would do on Android.

4.1.2.2 Android Support Library

4.1.2.3 Project Treble

4.1.2.4 Security updates

4.2 Kotlin

As discussed in chapter 4, Kotlin has become a primary Android language in 2017. After having used this language for a first Android project last year, I decided to build all my subsequent Android application in Kotlin. The simplifications and reductions in code length provided by this language compared to Java make developing Android applications more enjoyable. It also helps avoid many runtime errors by catching many error-prone scenarios at compile time. In this section, we will go in further details in some of the advantages provided by Kotlin and how Google is encouraging developers to use Kotlin by introducing new Kotlin-specific features in the Android SDK.

4.2.1 Full interoperability with Java

4.2.2 Data classes

Java is often defined by its detractors as a very verbose language, requiring to write a lot of repetitive boilerplate code[3] for simple tasks. One of these tasks is the creation of classes with accessors and mutators for some of the class fields and overriding `equals` and `toString` methods. In Object Oriented Programming, we often have to write a lot of small classes just to match the Models in our applications. These are often referred as Beans or POJOs[4]. Kotlin introduces the concept of data classes[6] to simplify the implementation of these type of classes. By using a data class, you get "for free" an

---

[3]Boilerplate code or boilerplate refers to sections of code that have to be included in many places with little or no alteration[5]

[4]Plain Old Java Object

accessor (and mutator) for each field of the class, a correct overriding of the `equals` method, an overriding of the `toString` method listing the values of all fields in the instance and a `copy` method corresponding to a copy constructor in Java. For example, if we take a simple person class in Java:

```java
class Person {
  private String firstName;
  private String lastName;
  private int age;

  public Person (String firstName, String lastName, int age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }

  public Person (Person other) {
    this.firstName = other.firstName;
    this.lastName = other.lastName;
    this.age = other.age;
  }


  public String getFirstName () {
    return this.firstName;
  }

  public String getLastName () {
    return this.lastName;
  }

  public int getAge () {
    return this.age;
  }

  @Override
  public boolean equals (Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Person person = (Person) o;
    return age == person.age && firstName.equals(person.firstName) &&
       lastName.equals(person.lastName);
  }
```

```
38
39    @Override
40    public String toString () {
41      return "Person{" +
42          "firstName='" + firstName + '\'' +
43          ", lastName='" + lastName + '\'' +
44          ", age=" + age +
45          '}';
46    }
47  }
```

And then the same class written using data classes in Kotlin:

```
1  data class Person(val firstName: String, val lastName: String, val
   ↪  age: Int)
```

4.2.3   Constants first

4.2.4   Class extensions

4.2.5   Null safety

4.2.6   Android KTX

4.3   Android Jetpack

4.3.1   LiveData

4.3.2   Room - Data persistence

4.3.3   Work Manager - Background jobs

4.3.4   Navigation

Chapter 5

NoSQL Database - MongoDB

5.1 Storage as JSON/BSON documents

5.2 How to handle references

5.3 Mongoose

5.3.1 Schemas

5.3.2 Promises with Mongoose

5.3.3 Populating references

5.3.4 Hooks

Chapter 6

NodeJS et Express

6.1 JavaScript ES6

6.2 NPM Modules

6.3 Promises

6.4 Middlewares

6.5 PassportJS

Chapter 7

Angular 6

7.1 Single-page webapp

7.2 Dependencies injection

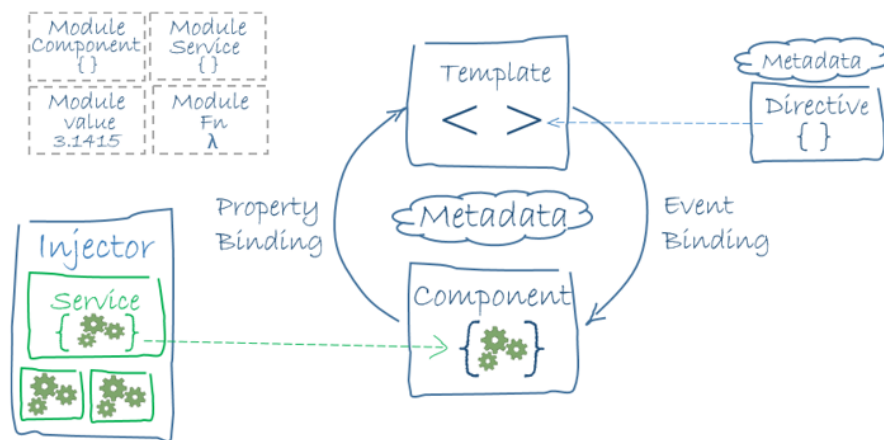## 7.3 Architecture of an Angular Application

Figure 7: Architecture of an Angular application[1]

7.3.1   Components

7.3.1.1   Templates

7.3.1.2   Communication between components - Input/Output

7.3.2   Services

7.3.3   Routing and Guards

7.4   RxJS - Observables

7.5   Reactive Forms

7.6   TypeScript

7.7   Angular Material

Chapter 8

Containers - Infrastructure as code

8.1 Docker

8.2 Orchestration avec Docker Compose

8.3 Google Cloud

8.3.1 Kubernetes

8.3.2 Google Container Registry

8.4 Azure

Chapter 9

Implémentation

9.1 Modules du projet

9.2 Déploiement

9.2.1 Architecture du projet

9.2.2 Proxy Nginx

9.3 Le backend commun

9.3.1 Authentification

9.3.2 Endpoints

9.3.3 Interaction avec la base de données

9.4 Le frontend Angular

9.4.1 Le routage

9.4.2 Les composants

Chapter 10

Résultats

Chapter 11

Conclusion

Bibliography

[1] Angular - Architecture overview. `https://angular.io/guide/architecture`. (Accessed on 23.06.2018).

[2] Camipro | EPFL. `https://camipro.epfl.ch/page-6801-en.html`. (Accessed on 21.06.2018).

[3] EPFL — PocketCampus. `https://pocketcampus.org/epfl-fr/#epfl-support-fr`. (Accessed on 22.06.2018).

[4] Distribution dashboard | Android Developers. `https://developer.android.com/about/dashboards/`. (Accessed on 07.06.2018).

[5] Boilerplate code - Wikipedia. `https://en.wikipedia.org/wiki/Boilerplate_code`. (Accessed on 18.06.2018).

[6] Data Classes - Kotlin Programming Language. `https://kotlinlang.org/docs/reference/data-classes.html`. (Accessed on 18.06.2018).