Haute École de Namur–Liège–Luxembourg

IESN

DEVELOPMENT OF AN ANDROID SUPPORT APP FOR FIELD AGENTS

By

Andrew KEYMOLEN

Namur

Academic year 2015-2016

Project sponsor

Chantal BERTRAND

Supervisor

Thomas NYSSEN

A thesis submitted in fulfillment of the requirements for the degree of BACHELOR OF BUSINESS COMPUTING

Haute École de Namur–Liège–Luxembourg
IESN

DEVELOPMENT OF AN ANDROID SUPPORT APP FOR FIELD AGENTS

By

Andrew KEYMOLEN

Namur

Academic year 2015-2016

Project sponsor

Chantal BERTRAND

Supervisor

Thomas NYSSEN

A thesis submitted in fulfillment of the requirements for the degree of BACHELOR OF BUSINESS COMPUTING

The writing of this report wouldn't have been possible without the intervention of a significant number of people. Therefore, I'd like to acknowledge them here properly. First, Vincent MERCHIE, a very patient developer at Djm digital who manage to successfully share his passion in development with me without ever failing on answering to my numerous questions. The whole staff of Djm digital is to be acknowledge as well for welcoming me within their team during those enriching four months. Among them, Dominique MAES for allowing me to do this internship, Aurélie LESAGE for organizing it and Thomas NYSSEN for agreeing to become my supervisor. I'd like to acknowledge Chantal BERTRAND as well, my project sponsor and teacher, for her monitoring and the support she gave to me all along those fourteen weeks. Finally, I'd like to thank all the people who helped and supported me directly or indirectly during my years of study.

TABLE OF CONTENTS

1 INTRODUCTION

As a professional activity of integration, students in their final year of business programing at Hénallux are being asked to do a fourteen weeks internship at a company. The goal of it is for the student to familiarize with the business landscape and for him to acquire a first experience in the professional world. On top of that, the student is asked to provide a thesis, i.e., a report explaining the internship in details as well as the different projects made during this period of time.

Therefore, I consulted several offers for a professional activity of integration before applying to the most interesting ones to my mind. After a few successful interviews, I ultimately chose a very actual field accordingly to my ambitions. Indeed, Djm digital was offering to welcome a student for him to be able to train to mobile development and they were the ones I ultimately chose.

The internship thus started in February and my mission consisted in the making of a support app for field agents named Ethias Prevention for the company Ethias. I was lucky enough to be the only developer working on it and, without denying the significant help I was given, it allowed me to grasp the nature of all its functionalities. They concern the making and centralization of risks reports. Therefore, an Ethias employee can use the app on the field to save its observations and access the result later on another device. The main focus of this thesis is this app which I will further analyze and explain in details.

After a brief presentation of Djm digital, a chapter will be devoted to a deeper description of the internship. Afterwards, I will enumerate the different tools and technologies used all along those fourteen weeks, e.g., the essential Android Studio.

The rest of the report will be devoted to a detailed study of the platform at which Ethias Prevention is aimed, i.e., Android. I will explain the history of this operating system as well as its working, the paradigms associated with it and the details of a development on it via an explanation of Java for Android and of the Model View ViewModel, the design pattern used.

Finally, and to conclude on the experiences lived during this internship, will follow the analysis and the presentation of the project as well as a reflection on the possible inputs for the development of such an application.

2 PRESENTATION OF DJM DIGITAL

The professional activity of integration and the project development proceeded as planned in a company at a rate of thirty-eight hours per week for fourteen weeks from February to May. Here is the presentation of this enterprise, Djm digital.

2.1 GENERALITIES

Figure 1 : Djm digital logo

78 Porte de Lorette

4600 Visé, Belgium

+32 (0)4 379 69 97

info@djmweb.be

http://www.djmdigital.be/

Private limited liability corporate entity based in Visé, Djm digital was founded in 1999 by Dominique MAES and grows since step by step. Today consisting of sixteen people from disciplines such as web, mobile, development and graphic design, Djm digital aims to advise clients, both upstream and downstream of their projects, for them to integrate the digital to their overall business strategy.[1]

2.2 LOCATION

The premises are located in Visé, capital of the Basse Meuse and recognized tourist center, halfway between the cities of Liege and Maastricht.

They are easily accessible by car from the highway by taking the exit to Visé or by public transportation, either by train or by bus from Liège or Maastricht, Visé having a train station.

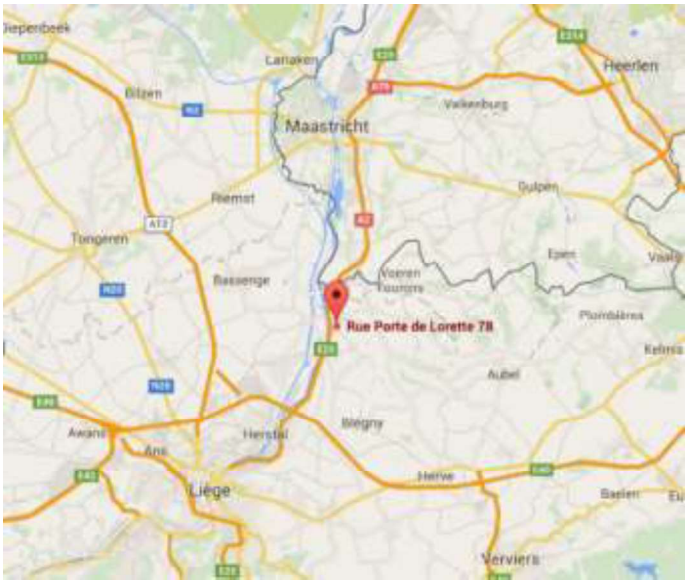

Figure 2 : location of Djm digital
(obtained on : https://www.google.be/maps/)

[1] (Djm Digital : Création de site internet & application mobile, s.d.)

2.3 INTERNAL ORGANIZATION

The main types of activities are mobile development, web development and e-marketing and consultation in each of these areas. These disciplines interact organically, they are organized around four poles, i.e., the design/graphics department, the development department, the project management department and finally, the administration department.

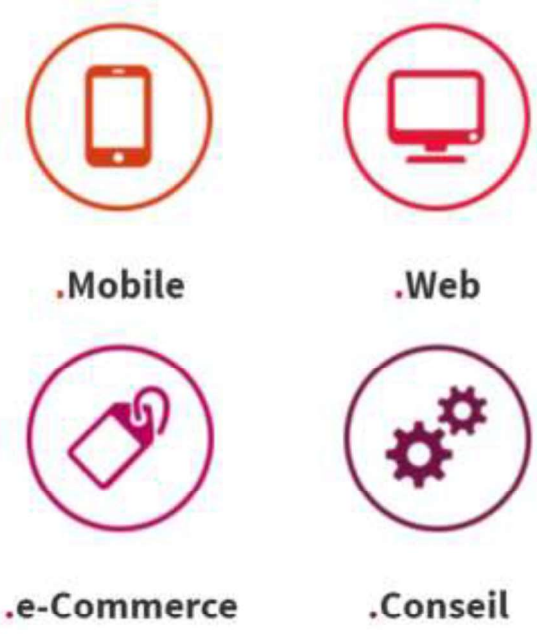

Figure 3 : the four fields of expertise at Djm digital

3 PRESENTATION OF THE INTERNSHIP

Started on Monday, February 8, 2016, this internship aims at the development of mobile apps for Android. If the first day is dedicated to the introduction to the different tools and technologies used at Djm digital, the first two weeks are devoted to a learning phase on an application whose nature was arbitrarily decided by Vincent MERCHIE, one of the Android developers at Djm digital.

The introductory task consisted in the development of a profile management app for T411, a website dedicated to the sharing of torrents, i.e., files containing the metadata of other files and folders made available for sharing by other users.[2]

This exercise served as a perfect training on the development for Android and covered most of the basic aspects. The following thirteen weeks are devoted to the main focus of the internship, an app ordered by Ethias aimed at its field agents. Once at the customer's place, the app must assist in drafting reports. Consisting mainly of forms to fill out, this application can also record photos, videos or audio messages as well as the place of intervention, legislations, pictograms and everything the agent would need to describe the situation he is attending to. Afterwards, these reports are centralized and can be accessed as needed from the application through a history.

The features of the app have been specifically agreed in advance - an iOS version of it being already available as well as a first attempt at implementing some designs. The internship is therefore mainly devoted to the completion and to the upgrade of the latter, with taking into account the new Android technologies, namely the Data Binding library and the MVVM design pattern.[3]

This task, and therefore the subject of this internship is conducted in the premises of Djm digital in a corporate environment, on a PC running Windows Seven, mainly with the help of Android Studio[4] and at a rate of about eight hours per day.

[2] (Torrent file - Wikiwand, s.d.)
[3] cf. 5.7.11 MVVM and data binding
[4] cf. 4.1 Android Studio

4 PROCESSES, TOOLS AND TECHNOLOGIES

Whether they're about organization, development support, centralization or others, the following tools were all needed during the internship and the development of the associated project.

4.1 ANDROID STUDIO

http://developer.android.com/sdk/index.html

Figure 4 : Android Studio logo

Android Studio is an IDE - Integrated Development Environment - which aims to provide a tool facilitating the creation and development of Android apps. Based on IntelliJ IDEA, a very complete commercial Java[5] IDE developed by JetBrains, it is suggested by Google in an "Early Access Preview" version in May 2013. This is to eventually replace a specific distribution of Eclipse, another Java IDE, provided with Android's SDK, i.e., the Software Development Kit. Eclipse is indeed the official solution previously offered by Google to the developers wishing to exercise on its operating system. In December 2014, Android Studio is released in 1.0 and is therefore recommended by Google to all the people who want to try the Android adventure.[6]

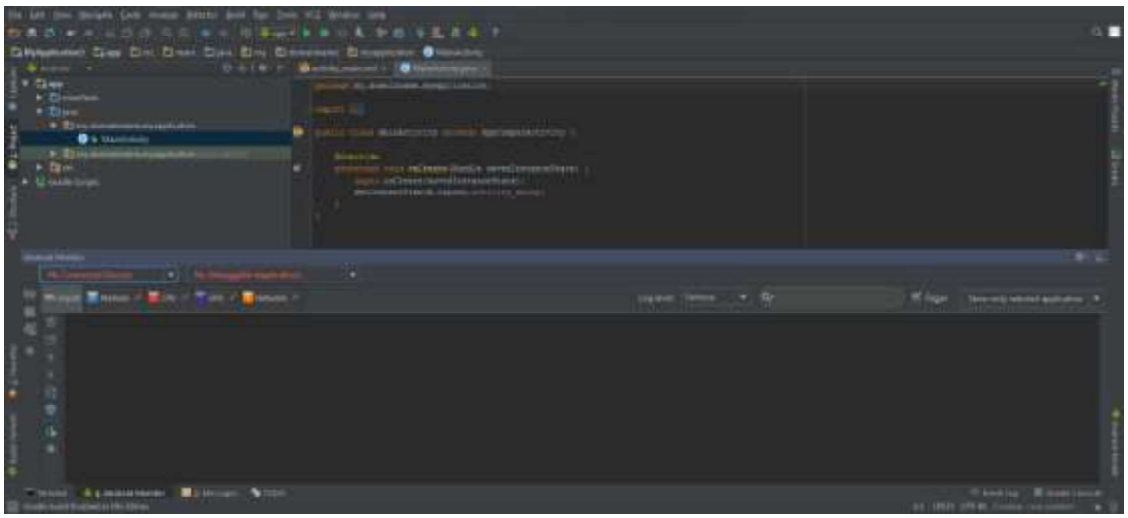

Figure 5 : user interface of Android Studio

[5] cf. 5.7.1 Java
[6] (Download Android Studio and SDK Tools | Android Developers, s.d.)

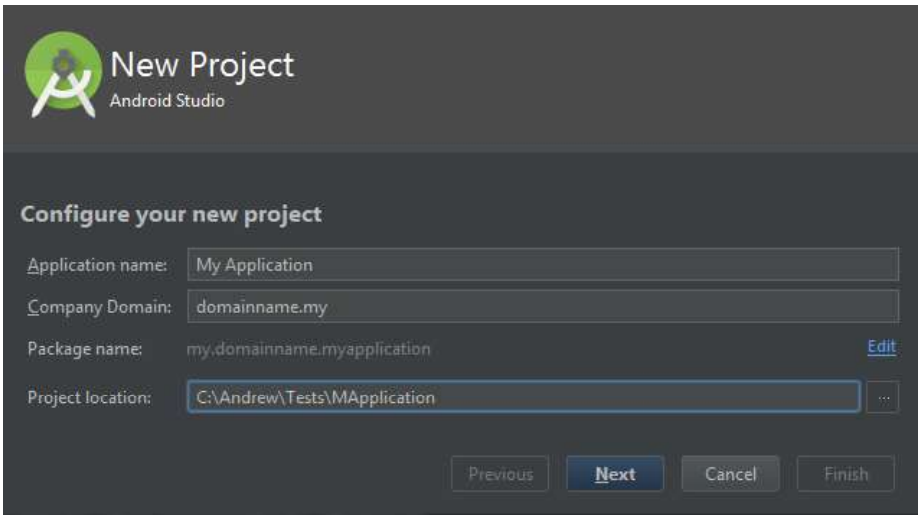

Figure 6 : « New Project » screen

In addition to its Java file editing function and to the configuration files needed for Android, Android Studio also provides tools to manage the development of multilingual apps and to directly visualize the appearance of pages created in various resolutions of screens.

The interface (see Figure 5 : user interface of Android Studio), rather standard compared to other IDE, lets you create a project in an intuitive way: going to "File" then "New" and "New Project" brings up a screen (see Figure 6 : « New Project ») for deciding the application name, the name of the company at which this application is developed, the name of the package in which the project is located and finally, the location of the project on the hard disk.

The next screen (see Figure 7 : « Target Android Devices ») makes it possible to decide the type of project to achieve. Android being used on different devices, a decision is needed here in views of the app compatibility. If the most frequent choice is the development for tablets and smartphones, it is also possible to target devices running Android Wear (the connected watches), Android TV (Smart TVs), Android Auto (compatible cars dashboards) and finally, Glass, the operating system of Google Glass - the augmented reality glasses. This screen is necessary to select one or more target devices, and

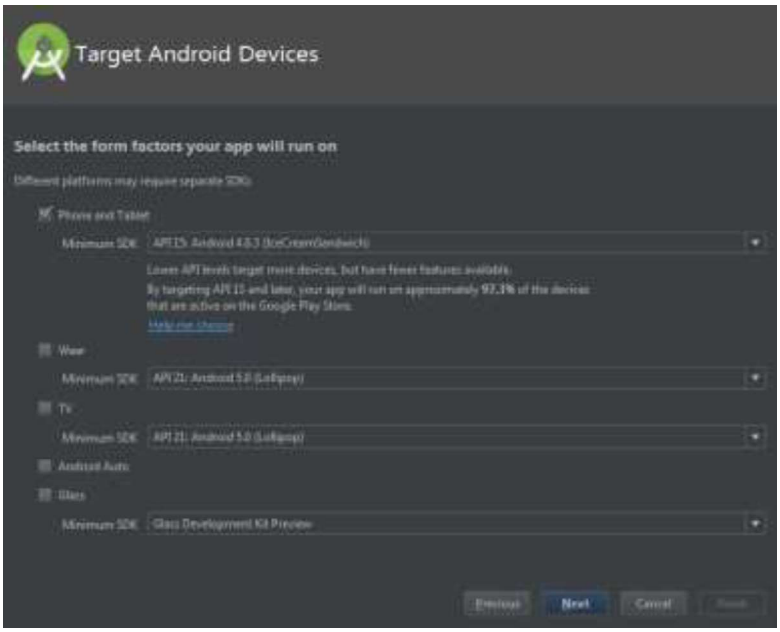

Figure 7 : « Target Android Devices » screen

therefore the audience, the corresponding API providing access to more or fewer features and backward compatibility. The rest of this presentation of Android Studio, however, is focusing on mobile development.

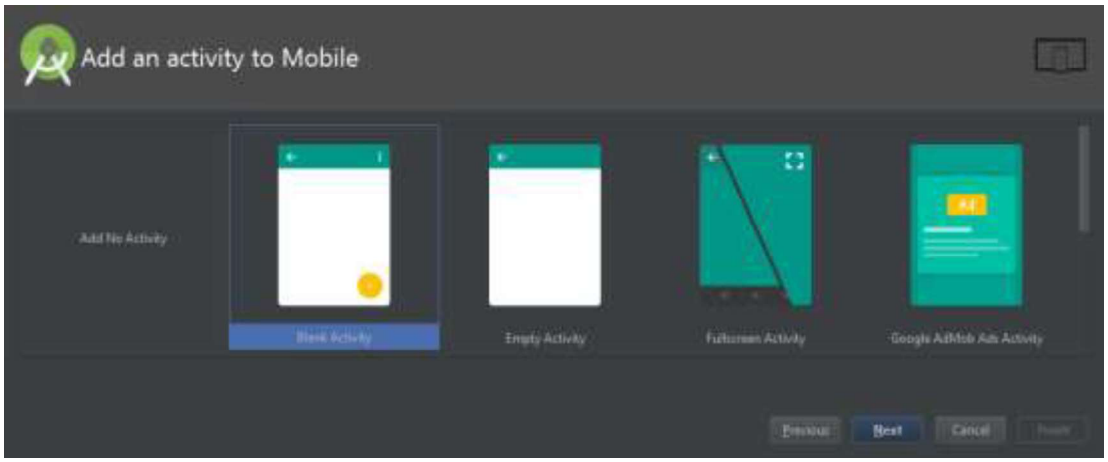

Figure 8 : « Add an activity to Mobile » screen

The next step focuses on the nature (see Figure 8 : « Add an activity to Mobile » screen) and the configuration (see Figure 9 : « Customize the Activity » screen) of the first activity,[7] i.e., the first screen being developed for the application. A good choice would be, for example, an empty or an almost empty activity or a login screen. Many other options are available and the choice belongs only to the developer.

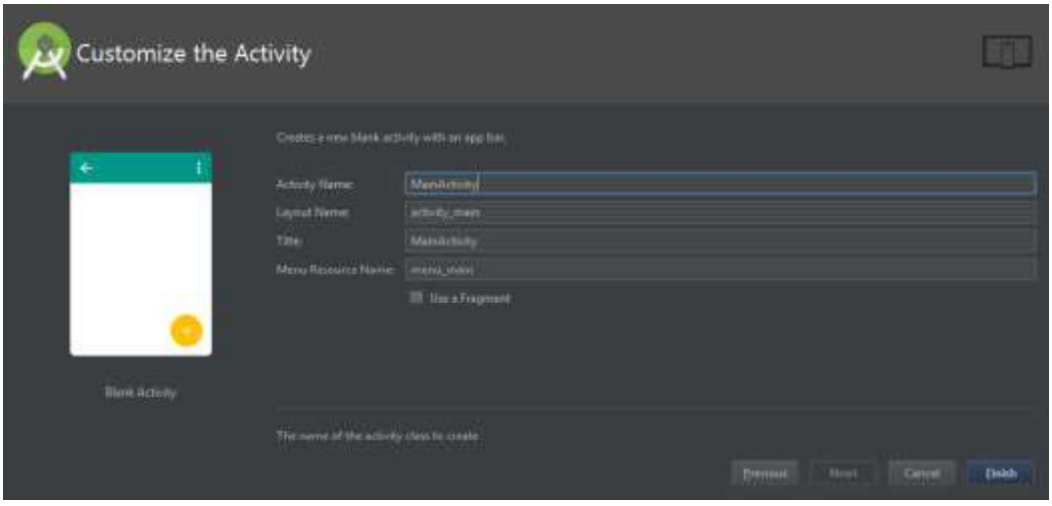

Figure 9 : « Customize the Activity » screen

Validating the first activity leads to the workspace (see Figure 5 : user interface of Android Studio) divided mainly into three parts. On the top left corner, the navigation area where the project structure[8] is. On its right, the editing area where the code is being written and below it, the console for displaying error messages, a Windows based

[7] cf. 5.7.4 Activity
[8] cf. 5.8.2 Structure

terminal and especially LogCat, a window in which it is possible to display customized messages through the "android.util.Log" class. This function is extremely popular among developers to debug the code very efficiently. LogCat makes it also possible to apply filters on what is being displayed to focus on one problem at a time.[9]

Finally, the toolbar at the top of the workspace provides access to all the necessary features such as compilation, execution or debug mode.

4.2 GRADLE

http://gradle.org/

Figure 10 : Gradle logo

The build system - the process that converts the code into an executable file - of Android is an Android plugin for Gradle. Gradle is an advanced production module for building projects in Java, Scala, Groovy and C ++. It manages dependencies – e.g., for libraries - and helps define a custom build logic. It is possible to use the Android plugin for Gradle independently of Android Studio, allowing the conversion of projects into executable files using command lines on machines where Android Studio is not installed, e.g., on a server. [10]

Written in Groovy, building files generated by Gradle make then possible the import of standard tasks, allowing sometimes to build programs using one or more languages, sometimes to import libraries into a .jar file, sometimes to run unit tests, etc., all remaining simple and compact.[11]

The build.gradle file allows easy configuration of the following aspects of the app during the development:

- **Construction variants**

It is indeed possible to generate different .apk - packages of compressed files allowing to launch the application on the mobile - from the same project.

[9] (DUBISY, 2015)
[10] (Android Pugin for Gradle | Android Developers, 2016)
[11] (MAZELIER, 2011)

- **Dependencies**

This concerns the management of libraries. No need to download any .jar files, Gradle takes care of it.

- **Inputs of the manifest**

It is possible, thanks to the build.gradle file, to vary the values of the manifest[12] - the file used to define the properties of the application – e.g., the project name.

- **Signature**

Gradle is responsible for signing the .apk file prior to the publication.

- **ProGuard**

The latter is a tool used to reduce and optimize the code. It is easily configurable in the build.gradle file.

- **Les tests**

Gradle creates an .apk dedicated to test the app.

Of course, when creating a project, all these aspects are already provided by default and must be determined only if necessary. Below, an example of a build.gradle file:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    dataBinding {
        enabled = true
    }

    defaultConfig {
        applicationId "com.andrew.fictional_login"
        minSdkVersion 15
        targetSdkVersion 23
```

[12] cf. 5.8.3 AndroidManifest.xml

```
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
    compile 'com.android.support:design:23.3.0'
}
```

4.3 OTHER TOOLS USED IN ADDITION

Rather secondary in views of the development of the main project, the tools briefly presented here are specific to the modus operandi of the developers within Djm digital. However, their respective functionalities, focused on project management, centralization and decentralization of the work being done, the test of this work and others, are powerful tools and it seems necessary to list them.

- **git**

Figure 11 : Git logo

https://git-scm.com/

A decentralized software used to manage project versions.

- **SourceTree**

Figure 12 : SourceTree logo

https://www.sourcetreeapp.com/

A Git and Mercurial desktop client aimed at developers facilitating the usage of Git.

- **SoapUI**

https://www.soapui.org/

Figure 13 : SoapUI logo

An open source application facilitating the testing of web services within a service-oriented architecture.

- **Node.js**

https://nodejs.org/

Figure 14 : Node.js logo

A free JavaScript software platform oriented toward network applications and allowing, e.g., to run a web server without the need for external software.

- **toggl**

https://toggl.com/

Figure 15 : toggl logo

A service facilitating the quantification of time spent to develop a project for billing purposes or to help estimate the time required for the realization of a new project.

- **Jira**

https://fr.atlassian.com/software/jira

Figure 16 : Jira logo

A bug tracking system, an incident management system and, more generally, a project management system.

- **Bitbucket**

https://bitbucket.org/

Figure 17 : Bitbucket logo

A web hosting service and a development management software similar to SourceTree, also using Git and Mercurial.

4.4 Equipment used during the tests

The project being designed for mobile devices, it becomes important to test it on several phones with different versions of Android to ensure the app behavior on each of them. The multiple screen sizes, the software layer added by the manufacturer or the device performance are to be considered as well. Here is the list of the various equipment used during the testing phase as well as their description.

- **Sony Xperia M2**
  - 4,8 inches 960x540 px
  - 1 GB
  - 1,2 GHz
  - Android 5.1.1

Figure 18 : Xperia M2

- **Nexus 7**
  - 7,02 inches 1920x1200 px
  - 2 GB
  - 1,5 GHz
  - Android 6.0.1

Figure 19 : Nexus 7

5 Android

Figure 20 : Bugdroid, the mascot representing Android

If there are three main operating systems that share the market in the mobile world, namely iOS, Windows Phone and Android, only the latter is open source and therefore allows the developer community to tackle its source code in order to get the best out of it. Perhaps one of the major reasons for its success, Google's policy - the developers of Android – in regards to its platform has undoubtedly transformed the mobile apps into what they are today and this transformation is not yet in its final phase.

Based on the Linux kernel, Android can be defined as a software stack organized in five layers:

- **Kernel**

The kernel, near the hardware, is responsible for drivers management.

- **Libraries**

The software libraries, useful to the different softwares that are installed downstream, facilitate the management of the necessary tools such as a database or graphics.

- **Android runtime**

The runtime allows you to run Java[13] programs - the language used to develop Android apps.

- **Application Framework**

The framework, which consists of a kit helping developers create apps.

- **Applications**

Standard applications like the web browser, the call function, etc.

[13] cf. 5.7.1 Java

Of course, it does not stop there as it is also worth noting that Android Studio and Google Play,[14] by their undeniable usefulness, are part of what is known as the Android features.

Originally intended to cameras, the system can now run all kinds of devices from washing machines to the last smartwatches including cars and televisions.[15] In 2015, it's Google mobile operating system that is on top with 80% of the market share in smartphones.[16] So it seems that Android is not at its last version and while its developers are already thinking about a tidbit name beginning with x, y or z,[17] others are still studying its current functionalities, being detailed further below.

5.1 History

A platform with such a great influence today deserves from us we come back to what made it most mobile developers number one priority. Android's story begins in 2003 when the eponymous startup is founded. If few details sweat about this startup, it is still common knowledge that it is a former employee of Apple, Andy Rubin, who founded it and that he always intended to develop an operating system for mobile devices. But in 2005, Rubin agrees to sell Android Inc. to Google for an estimated fifty million US dollars and becomes vice president of engineering at Google in order to oversee the development of its creation.[18]

If someone else is now seated at the place of VP, the fact remains that the platform evolves much under the wing of Rubin. In 2006, the decision is taken to adapt Linux OS into something less archaic and more capable of serving a mobile user. But it's in 2007 that real change occurs. In November of this year, the OHA - Open Handset Alliance, a group of companies whose goal is to create open source mobile standards in order to give a boost to the sector - is officially announced. The result that follows is the one we know. In 2008, the first device running Android is released, thus starting a long run to new versions, innovative features and better performance. The very latest version, announced in May 2015, is Android 6.0 - Marshmallow - but Android N, the next version of which an overview and the development tools are already available, should see its official release date revealed soon.[19]

Google however pleaded a change in strategy recently. If until now Android is primarily for mobile and, by extension, for tablets, the advent of Android in other

[14] cf. 5.3 Google Play
[15] (DANON, 2014)
[16] (ZDNet.fr, 2015)
[17] cf. 5.2 Android's different versions and their APIs
[18] (LEGGETT, 2011)
[19] (LARDINOIS, 2015)

devices can now be observed. An announcement in March 2014 reveals Android Wear,[20] an operating system for connected watches. If this announcement is not surprising for the press, the following two that take place in June of the same year are a little more. Indeed, it is at its annual meeting with the public, the Google I/O, that Google announced Android TV,[21] an operating system designed to invade living rooms, as well as Android Auto,[22] a projected system allowing drivers to control their Android smartphone using only the different controls of the car.

Anyway, Android still has a long life ahead of him even if it starts becoming multidisciplinary. Developers can only hope that Google continues to support and help them with the upcoming new functionalities of this so versatile and accessible OS.

5.2 Android's different versions and their APIs

As explained previously, Android has gone through many versions, each bringing a lot of new features. If Marshmallow - Android 6.0.1 - is the latest version, it is with Android 1.1, apparently named Petit Four by the time,[23] that it all starts with regard to the consumer. It is interesting to note that from version 1.5, or Cupcake, each version is given the name of a candy in alphabetical order, the previous version undergoing the same treatment later to match this logic. A number of them still being in circulation, it is good to keep it in mind when developing an app to determine the target audience and therefore to choose the API to use. For example, KitKat, or Android 4.4, is, at the time of writing this paragraph, the more present version of Android on the market.[24] Therefore, it is currently fashionable when developing an app to make it Kitkat compatible.

The following is a list of the major versions of Android and their corresponding version of the API:

Version	Code name	API	Release year	Main features
1.0	*Apple pie*	1	2008	Youtube app Google Maps

[20] (D'ORAZIO, 2014)
[21] (OPAM, 2014)
[22] (BANERJEE, 2014)
[23] (AMADEO, 2012)
[24] (Dashboards | Android Developers, 2016)

				Gmail Synchronization, Contacts and Google Agenda Camera feature Web browser Downloads and updates via the Android Market Place[25]
1.1	*Petit Four then Banana Bread*	2	2009	Save functionality for the MMS attached files
1.5	*Cupcake*	3	2009	Bluetooth support Touch keyboard Video support
1.6	*Donut*	4	2009	Framework for tactile functionalities
2.0	*Eclair*	5 to 7	2009	Interface enhancements
2.2.x	*Froyo*	8	2010	Apps installation on the external memory Upload functionality in the web browser
2.3.x	*Gingerbread*	9 and 10	2010	Interface and ergonomics enhancements Social features Video call
3.x.x	*Honeycomb*	11 and 13	2011	Multicore support Improved stability on tablets Introducing the Action Bar and the Fragments

[25] cf. 5.3 Google Play

4.0.x	*Ice Cream Sandwish*	14 et 15	2011	New lock screen Web browser and speech recognition enhancements Facial recognition
4.1.x	*Jellybeans*	16	2012	Google Now, a smart personal assistant Voice search Accessibility features
4.2.x		17	2012	Lock screen widgets Several sessions possible on tablets
4.3.x		18	2013	4K support
4.4.x	*KitKat*	19	2013	Translucent interface Better notifications
5.0.x	*Lollipop*	21	2014	Material design[26]
5.1.x		22	2015	Call HD
6.0	*Marshmallow*	23	2015	Android Pay, a card free payment system Better permissions management Fingerprint reader support

Figure 21 : table of the different Android versions[27]

[26] cf. 5.6 Responsive design and material design
[27] (Android versions comparison | Comparison tables - SocialCompare, 2016)

Here is the distribution of the various versions of Android on the market as of February 1, 2016.[28]

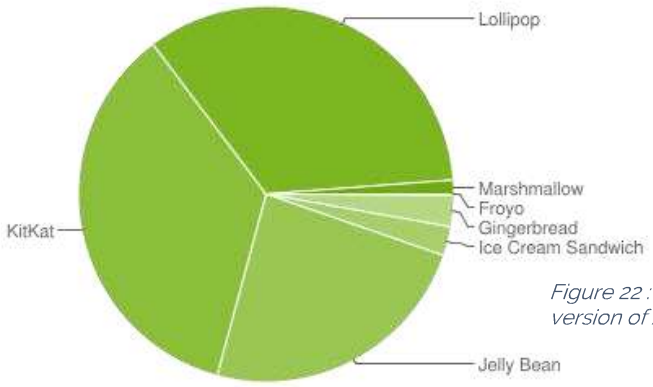

Figure 22 : distribution of the various version of Android on the market

Lollipop, KitKat and Jellybeans are unsurprisingly the three most popular versions of Android and therefore those to target when developing an application. There is also Marshmallow, still very new but very promising, whose compatible devices are only coming.

5.3 Google Play

Google Play https://play.google.com/

Figure 23 : Google Play logo

Huge online library of free and non-free apps, music, movies and books, Google Play, formerly Android Market, is the online store where each Android user can search and download what he desires and provide feedback for each of its purchases. The buyer can return to the page of an application to leave an evaluation, a comment or update this app.

This service is also useful for developers, they can publish an app with minimal effort. Indeed, all they need to do is to export the app on Google Play, certify it and then send it to the store.

[28] (Dashboards | Android Developers, 2016)

5.4 Android Software Development Kit

The Android SDK is a set of tools used to facilitate the creation on this platform. If Android applications are developed in Java,[29] it is actually a variant of this language that a device with Google's mobile OS speaks. This is where the SDK becomes useful, it facilitates the writing of Android oriented Java by ensuring that Android Studio, the official IDE for the Android SDK, can best assist developers.

The altruism of Google towards the developers, however, is not enough. Java is the programming language used here, therefore, it's first necessary to install the Java environment, the JDK - Java Development Kit.

The SDK provides a wealth of practical tools such as a debugger, software libraries, an emulator to simulate different versions of Android and documentation, code samples and tutorials. Finally, each version of Android SDK also contains the complete source code of Android.

Available to all on the official website of the Android developer guide,[30] its interface (see Figure 24 : Android SDK) is intuitive and lets you choose which API to install before the development.

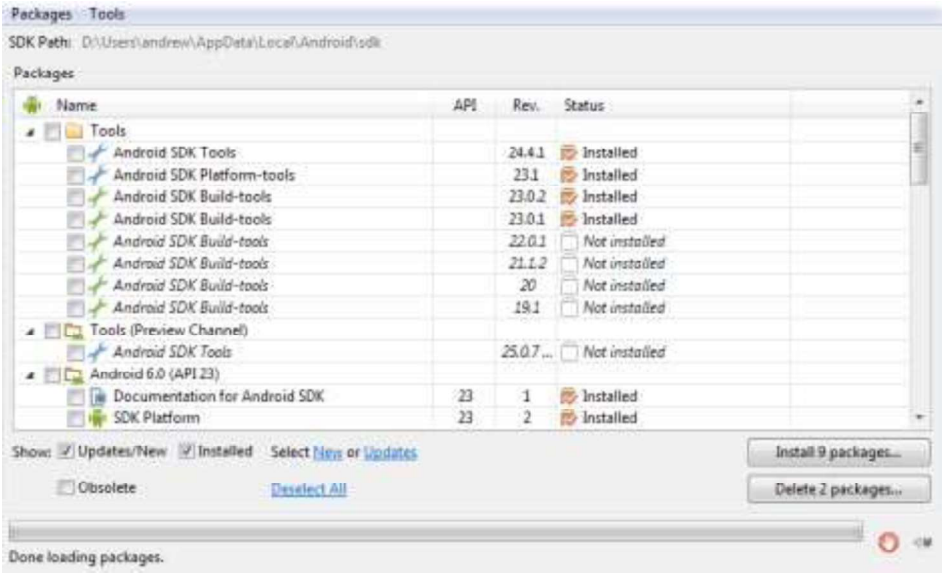

Figure 24 : Android SDK user interface

[29] cf. 5.7.1 Java
[30] (Download Android Studio and SDK Tools | Android Developers, s.d.)

5.5 ARCHITECTURE

The Android architecture is presented here by its developers as organized into five layers, from highest to lowest[31]:

Figure 25 : the Android architecture according to the Android Open Source Project, part 1

- **Application framework**

The application framework layer, dedicated to developers, provides APIs specific to the various features of the phone. Above is the user layer but since it's determined by the different manufacturers, it is omitted in the presentation of the Android architecture.

- **Binder IPC proxies**

The Binder IPC layer - Inter-Process Communication - provides the framework used to bypass the limits of the process in order to contact the code of the Android system services. This layer is invisible to the developers but it's this one that allows communication between the applications and the features of the phone.

- **Android system services**

The system services layer is the link between the framework and the hardware of the phone. It concerns the access to features that are organized modularly within two broad categories: the "system" part, which handles notifications, windows and others; and the "media" part dedicated to the recording or the playback of different media.

[31] (Android Interfaces and Architecture | Android Open Source Project, 2016)

- **HAL**

HAL - the Hardware Abstraction Layer - defines a standard hardware interface that manufacturers can implement, allowing Android to not worry about the hardware it runs on. It's the layer aimed at the different mobile manufacturers.

Figure 26 : the Android architecture according to the Android Open Source Project, part 2

- **Linux kernel**

The Linux kernel layer contains all of the device drivers. Here lies the last bridge between hardware and software. Its name comes from the fact that Android is based on a Linux kernel optimized for mobile.

5.6 Responsive design and material design

If Android is the most popular mobile OS, it is certainly to a large extent because it is compatible with the biggest number of devices. What makes it possible is its willingness to adapt and therefore its flexibility with regard to different display sizes. Such a feat is possible because Android categorizes the different screens according to two properties: the size and density. Each of these properties can take four values: small, normal, large and xlarge for the size and low (ldpi), medium (mdpi), high (hdpi) and extra high (xhdpi) for the density.

Therefore, based on these standards, developers can count on a relative measurement unit, the dp or dip for Density-independent Pixels, thereby maintaining the proportions of the resources regardless of their arrangement on the screen. There's another unit as well, i.e., the sp, for Scale-independent pixels, but this one is used for the fonts, allowing the latter to adapt to the user's preferences.[32]

Consequently, it's necessary to create different resources depending on the screen categories and to arrange them with the help of the visualization tools offered by Android Studio. This remarkable permissiveness gives rise to an adaptive design, or a design that reacts to the different screen sizes and to the device orientation.

[32] (Supporting Multiple Screens | Android Developers, 2016)

To go further and to ensure the ergonomics of the projects on Android, Google has decided to publish a guide on material design.[33] Adaptive and intuitive design of their creation, material design is a collection of rules, a standard to which each developer can refer to get a practical app that requires no explanation about how to browse it.

The aim of material design is to embody a visual language that merges the good design practices in general with the innovation and the opportunities now offered by technology and science. The result is a coherent interface respecting the physical spaces in three dimensions and providing a rich user experience. Therefore, the user can intuitively experiment with the virtual space embodied by the touch interface of the app, guided by rules focused on depth, light and shadow, movement and animations of the different elements and their interactions with each other within this space.

Figure 27 : the two states of a button according to the material design

Therefore, and for illustration purposes, a tile casting a shadow on an underlying element seems detached from its support, indicating a possible interaction. Pressing this tile enlarges its shadow because of the increase of the virtual distance between the tile and the underlying element. This button is therefore enabled, as its shadow testifies, and stay like that until a new interaction between it and the user (see Figure 27 :).

Of course, this is only one example among many rules. Applying them is not complex and gives a very satisfactory result. Also, certain elements in Android Studio such as the Action Bar or the Navigation Drawer already take the material design into account. A reminder of these guidelines can be found in Google's guide available online.[34]

[33] (Introduction - Material Design - Google design guidelines, s.d.)
[34] (Introduction - Material Design - Google design guidelines, s.d.)

5.7 Developing on Android

Knowing what Android is and what it offers is indeed interesting but not sufficient from a developer's point of view. That is why this part is focused on what an Android app is in terms of implementation. Without diving into code, this part linger on the explanation of the terms, concepts and paradigms of Android oriented Java.

5.7.1 Java

http://www.oracle.com/technetwork/java/index.html

Figure 28 : Java logo

If there is something without which Android would not be, it is its implementation language. Java is a particularly popular language introduced in 1995 by John Cage, director of the Science Office at Sun Microsystems, and Marc Andreessen, co-founder and executive vice president of Netscape.[35] In 2010, Oracle bought Sun and became the official owner and developer of Java.[36]

Java is an object oriented programming language whose characteristic and goal is the portability of the programs written in this language. Such a software can be ported to various operating systems such as Windows, UNIX, Mac OS and GNU/Linux with little or no modification. Being object oriented means that the data types of a project must be associated with the different operations that can be performed on those data. This association is made in an entity named "class". This class contains the "objects", i.e., structures that define the data when their properties vary, and their associated methods, i.e., the multiple operations implemented for those objects.

That's the main paradigm of the language, introducing other directly related concepts such as inheritance and polymorphism. The former enables reusability and adaptability of objects using the latter which provides a unique interface to the different types of entities. As for the rest of it, like any other programming language, Java also has its own reserved words, its own types and its own approach to the various control structures.

[35] (BYOUS, 2003)
[36] (Oracle and Sun Microsystems | Strategic Acquisitions | Oracle, s.d.)

5.7.2 Application class

The Application class is the base class of each app used to maintain its overall state. Each app has a default one and instantiates it as soon as it starts.[37] It is possible to implement this class, to override it or to implement a class that extends it in order, for example, to declare global constants or to persist information, e.g., certain objects or data and some user preferences. This class also provides access to the global context[38] of the app.

5.7.3 Context

An app context is an interface providing access to global information related to its environment.[39] In practice, this context is accessible from any activity by the method "getContext()" or "getApplicationContext()". It is also accessible through a call to the Application class. Allowing to link the app with the system, once called, the context can be used to start an activity or to send and consume an intent.[40]

5.7.4 Activity

An activity, essential to the development on Android, represents one of the views of the app and its interaction with the other activities and their potential inner fragments.[41] It usually concerns one of the use cases[42] of the app and is therefore responsible for the behavior of the graphical user interface.

5.7.4.1 Activity lifecycle

An important aspect of the activities and of the development on Android in general is the activity lifecycle. Indeed, as it's being used, an activity actually goes from one state to another. At each change of state, the system executes a method that the developer can override to implement the behavior the app must have when those changes happen.

[37] (Application | Android Developers, 2016)
[38] cf. 5.7.3 Context
[39] (Context | Android Developers, 2016)
[40] cf. 5.7.7 Intent
[41] cf. 5.7.5 Fragment
[42] cf. 6.1.3 Use case diagram

The diagram available on the Android Developers website[43] (see Figure 29 : the activity lifecycle according to the Android Developers website) perfectly describes this lifecycle.

- **onCreate()**

The onCreate() method is called at the creation of the activity and is used to initialize static variables, to create the views and to link[44] the data. This method also provides the backup of the previous state of the activity and is always directly followed by a call to the onStart() method.

- **onRestart()**

The onRestart() method is called when the activity restarts after it was stopped during a call to the onStop() method.

- **onStart()**

Directly called after onCreate(), onStart() marks the moment when the user can see the activity. It is followed by a call to onResume() or onStop() depending on whether the activity comes to the forefront of the application or not.

- **onResume()**

onResume() marks the moment at which the user can interact with the activity and can only be followed by a call to onPause().

- **onPause()**

Used to save the data that developers want to persist, to stop any animations or to pause the various processes using the device's resources, the onPause() method needs to be as light a method as possible because the activity can't resume before the end of onPause().

- **onStop()**

Called when the activity becomes invisible to the user due to a call to another activity or to the destruction of the current one, onStop() is followed by onDestroy() or

[43] (Activity | Android Developers, 2016)
[44] cf. 5.7.11 MVVM and data binding

by onRestart() depending on whether the current activity is destroyed or becomes active again.

- **onDestroy()**

Meaning the end of the activity, onDestroy() is called to release the resources before the application goes on to something else.

It is interesting to note that a configuration change, i.e., a change in the device orientation, also causes the destruction of the activity before its re-creation. Thus saving data to be stored during the execution of onPause() becomes important.

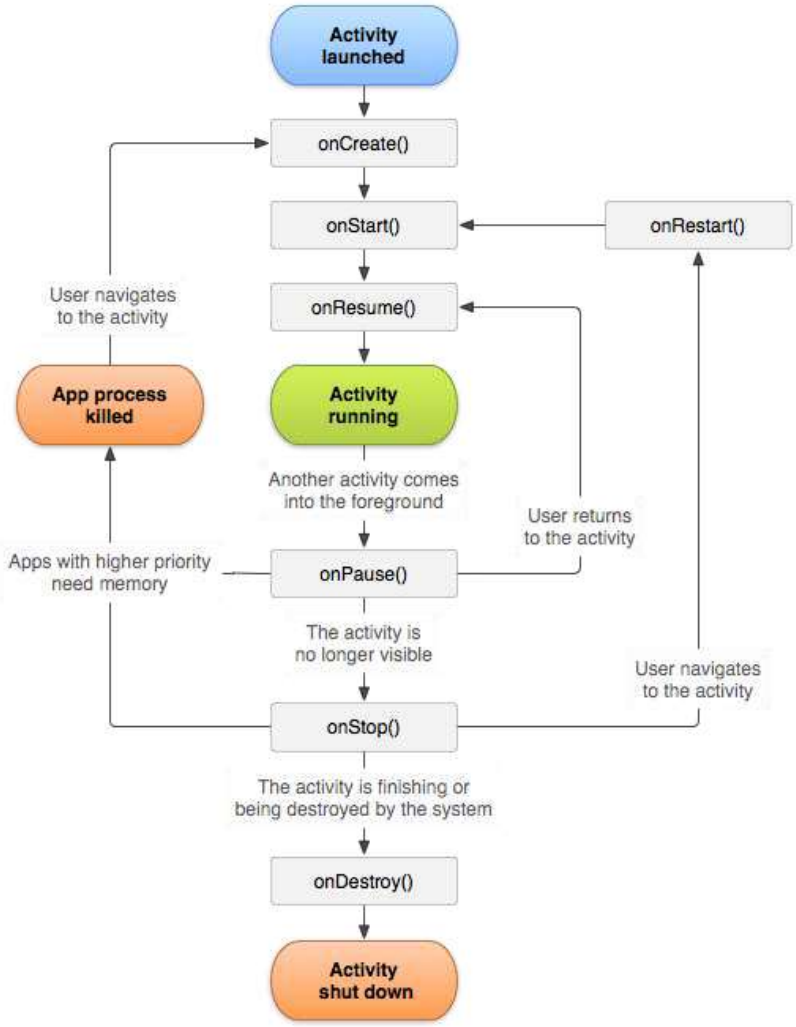

Figure 29 : the activity lifecycle according to the Android Developers website

5.7.5 Fragment

Introduced with Android Honeycomb, or Android 3.0,[45] and made backward compatible by the support libraries,[46] fragments are substantially similar to the activities, although a fragment can only exist within an activity. The fragments are used to create more complex interfaces, to modulate the code or to facilitate the work of adapting interfaces to different screen sizes,[47] very convenient for the development of smartphone apps as well as tablet apps.

It is interesting to consider the following methods as part of the fragment lifecycle (see Figure 30 : the fragment lifecycle according to the Android Developers website):

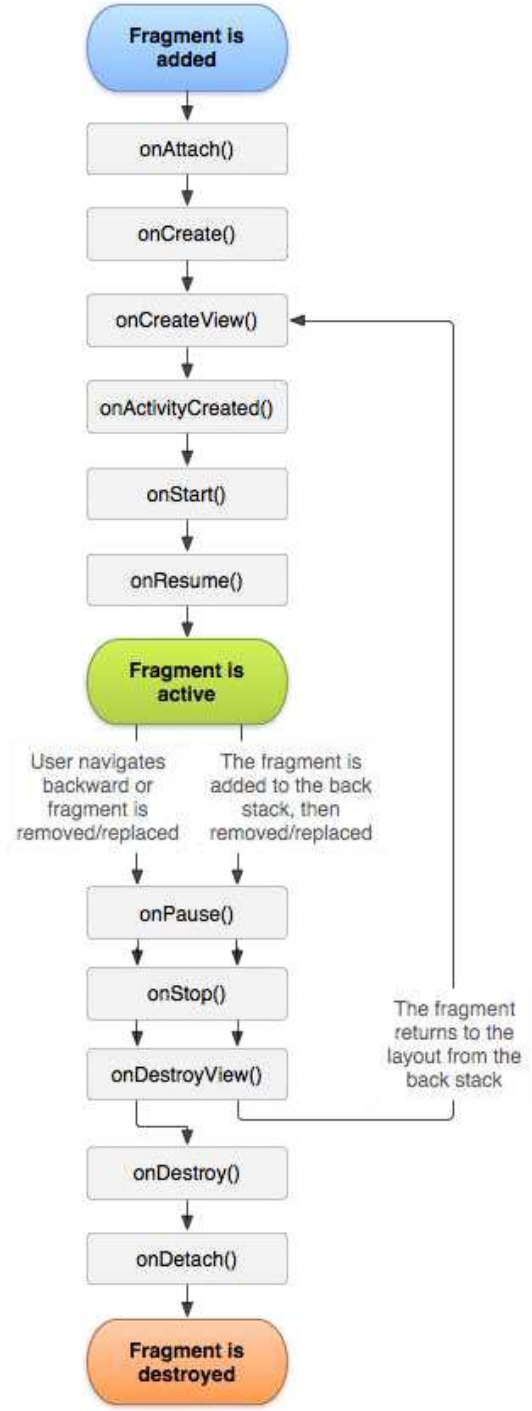

Figure 30 : the fragment lifecycle according to the Android Developers website

- **onAttach()**

onAttach() is called when the fragment is added to the activity and therefore before onCreate().

- **onDetach()**

onDetach() is called when the fragment is detached from the activity and thus after onDestroy().

- **onActivityCreated()**

This method is called when the activity is created, when the views of the fragment are instantiated. It is advisable to restore the state of the fragment within this method when necessary.

[45] cf. 5.2 Android's different versions and their APIs
[46] cf. 5.7.10 Android support library
[47] (Activity | Android Developers, 2016)

- **onActivityDestroyed()**

onActivityDestroyed() is called right before onDestroy().

5.7.6 Service

Similar to the activities and normally used for long tasks in the background like downloading, heavy data processing or audio file playback, services are independent of the user interface and thus continue their operations even when the application goes on to something else.[48] However, they have their own lifecycle and they can access the app context[49] if necessary.

5.7.7 Intent

An intent is an abstract description of an operation to perform and thus serves as a system message used mainly by the activities and the services.[50] The main information pieces of an intent are the action to perform and the data needed to perform this action. "Extras" are often added as well, i.e., additional data or information relevant to the operation that are to be performed.

The intents come in two forms:

- **Explicit intents**

Explicit intents have specific components that provide the exact class to run. The best example here is the intent for starting an activity, the class of the activity to display being specified in it.

- **Implicit intents**

Implicit intents, contrary to the explicit ones, do not specify the class to run. Thus, they must have enough information for the system to provide the adequate component to run. Sending an email from an app requires this kind of intent but it needs to sufficiently specify the nature of the process to run for the system to suggest to launch a relevant already installed application.

[48] (Services | Android Developers, s.d.)
[49] cf. 5.7.3 Context
[50] (Intent | Android Developers, 2016)

5.7.8 Gestures

The arrival of the touch screen in mobile telephony and especially in smartphones is a revolution in the field of interaction between man and machine that only the keyboard and the mouse are up to. Accordingly, the gestures are the options available to the user for communicating with the apps through the touch features of a device. If they now seem natural to almost everyone, it's because the developers had to think through the functions of each. They had to establish a standard and to implement and make the best use out of them to maximize the ergonomics of their software.

Now accomplished to the point that Google describes them in its guide on material design,[51] [52] these gestures embody a separate language where the motions to perform, called Touch Mechanics, would be the vocabulary and where their effects, the Touch Activities, would represent the grammar.

Here is a complete list of the currently existing Touch Mechanics with their name, a description of the corresponding motions and an example of an output - a Touch Activity - usually generated by this Touch Mechanic[53]:

Name	Description	Example
Touch	One-finger press, lift	Select
Double touch	One-finger press, lift, one-finger press, lift	Zoom in
Drag/Swipe/Fling[54]	One-finger press, move,[55] lift	Scroll
Long press	One-finger press, wait, lift	Select an element, such as a list item
Long-press drag	One-finger press, wait, move, lift	Pick up and move
Double-touch drag	One-finger press, lift, one-finger press, move, lift	Pick up and move

[51] (Introduction - Material Design - Google design guidelines, s.d.)
[52] cf. 5.6 Responsive design and material design
[53] (Gestures - Patterns - Google design guidelines, s.d.)
[54] The three are distinguished by the speed at which the Touch Mechanic is performed.
[55] A movement means dragging one or more fingers in one direction while remaining in contact with the touch surface.

Pinch open	Two-finger press, move outwards, lift	Zoom in
Pinch closed	Two-finger press, move inwards, lift	Zoom out
Two-finger touch	Two-finger press, lift	Zoom out
Two-finger drag/swipe/fling	Two-finger press, move, lift	Scroll
Two-finger long press	Two-finger press, wait, lift	None (uncommon gesture)
Two-finger long press drag	Two-finger press, wait, move, lift	Pick up and move
Two-finger double touch	Two-finger press, lift, two-finger press, lift	Zoom out
Rotate	Two-finger press, simultaneously orbit both fingers around the center point, lift	Rotate content

Figure 31 : table of Android's gestures

Given the importance of these features, Android comes with the classes and interfaces needed to facilitate their implementation. Touching a button initiates a touch event that the developer has to intercept through a listener from the OnClickListener class previously linked to the appropriate view. This listener implements the reaction that the app must have when such an event occurs. The system takes therefore care of detecting the event, determining its nature and triggering the appropriate behavior chosen by the developer.

5.7.9 ADAPTER

An Android adapter represents a concept that brings two elements together, namely an AdapterView and an Adapter object.[56] The former represents the views type, e.g., a ListView, a GridView, a RecyclerView or a Spinner that are different useful lists aimed at presenting the information to the user. The latter is the object that will prepare this information to display inside the former. Thus, by creating an instance of

[56] (Adapter | Android Developers, 2016)

Adapter and by passing the necessary information to it, the Adapter takes care of carrying out the necessary changes. The result is then reflected by the AdapterView, presenting the data as desired and taking care of the potential interactions with the user.

5.7.10 Android support library

With each novelty come problems. While each Android update brings its share of new features, each existing app can potentially become obsolete. Also, developing a new tool for a fleet of devices as large as possible while maximizing its modernity and therefore by implementing the latest Android features becomes a challenge. This is why Google provides what he calls the Support Library. This aims to make compatible the latest Android features with older versions of the operating system to prevent developers from having to implement these functionalities themselves with code that old versions of Android can understand. There are many versions of those libraries, each focusing on specific features or designed to run the latest apps on a particular version of the operating system.

The release dates of these libraries and the specifications of their different features being disclosed by Google in a rather disparate way and given their complex nature, it is not really easy or even very interesting to explain them in detail. However, given their undeniable usefulness, here are a complete list of those available today[57]:

- **v4 Support Library;**
- **Multidex Support Library;**
- **v7 Support Libraries;**
- **v8 Support Library;**
- **v13 Support Library;**
- **v14 Preference Support Library;**
- **v17 Preference Support Library for TV;**
- **v17 Leanback Library;**
- **Annotations Support Library;**
- **Design Support Library;**
- **Custom Tabs Support Library;**
- **Percent Support Library;**
- **App Recommendation Support Library for TV;**

[57] (Suport Library Features | Android Developers, 2016)

In practice, when adding a recent feature to the app code, Android Studio[58] spontaneously offers to import one of these libraries to make the project compatible with the largest number of devices possible. A line of code is therefore often enough to please the greatest number.

5.7.11 MVVM AND DATA BINDING

Google I/O is the annual conference at which Google announces its latest and upcoming projects. It was during this event in May 2015 that Google announced the Data Binding library,[59] a gift to every Android developers supposed to help decouple the user interface from the rest of the code. The different views of the application can therefore be linked to data without having to be referenced in the code manipulating that data, resulting in a much greater permissiveness with respect to potential downstream changes in the interface.

This library facilitates the implementation of the code according to the MVVM design pattern, the Model View ViewModel, an alternative to the popular MVP, the Model View Presenter, and to the MVC, the Model View Controller (see Figure 32 : MVVM, MVP and MVC).

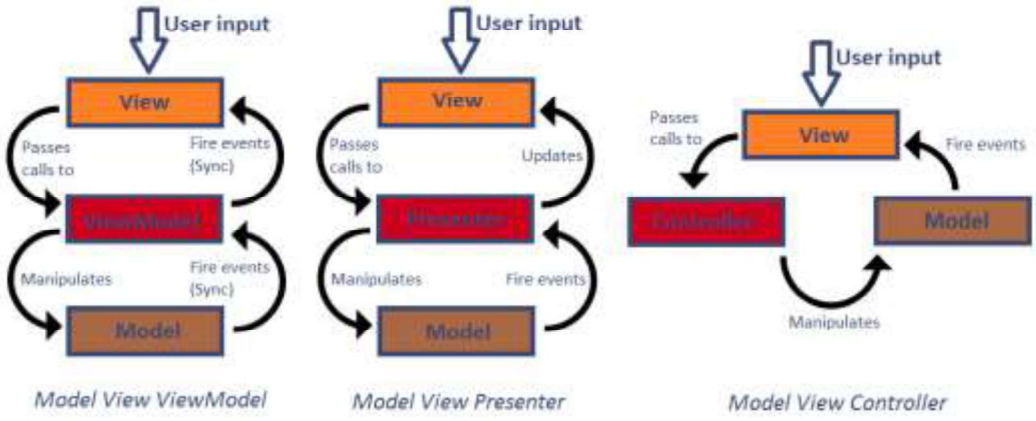

Figure 32 : MVVM, MVP and MVC

5.7.11.1 PRESENTATION OF THE MVVM DESIGN PATTERN

The MVVM design pattern is used to separate at best the GUI development from the business development of a program. It perfectly fits the mobile world since the

[58] cf. 4.1 Android Studio
[59] (Getting Started with Data Binding in Android, 2015)

interface is often highly complex and since the development requires some rigor in regards of the interactions with the user.[60]

This pattern is designed to divide the code into three distinct layers[61]:

- **Model**

This layer does not differ from the usual Model layer of other design patterns. This part of the code is devoted to the representation of data in a way fitting an object oriented language. If the representation of a user, for example, is necessary for the application and the user differs from others in his email address and his password, then a class "User" is implemented with two arguments, i.e., "mail" and "password". In this category are also available the getters and the setters of the arguments. They're used to obtain or modify their value. It is also customary to implement methods such the one used to represent the data as a String object, often called "toString()".

This layer also includes the services used to access remote data when using a database.

- **View**

Similar to the Model layer, the View layer is not very different from the usual View layer of the various design patterns. In Android's case, different views are implemented in XML, i.e., Extensible Markup Language - a markup language such as HTML[62] - to immediately establish the elements the user can access. It's also within the XML code that the ViewModel layer elements are linked to the View layer elements through data binding.[63]

The business code specific to the views - and therefore not bound to the data entered by the user, but rather to the interaction with him or to the size of the screen used[64] - is implemented in Java[65] in the same layer as well but in another file.

- **ViewModel**

The ViewModel layer is the most important layer of the MVVM pattern and the one that makes this pattern different from the others. It represents an abstraction of the

[60] cf. 5.7.8 Gestures
[61] (BIRCH, 2015)
[62] (WALSH, 1998)
[63] cf. 5.7.11.2 The Data Binding library
[64] cf. 5.6 Responsive design and material design
[65] cf. 5.7.1 Java

views that handles the communication between the Model layer and the View layer using notifications. The elements of the View layer are linked to those of the ViewModel through data binding. If a user action is to modify data, then the system must notify the relevant element in the ViewModel. The element is then modified as necessary to reflect any changes in the interface before a method contacts the Model layer to change and persist the data.

The user, by interacting with the interface, engages the functionality of the Data Binding library[66] which acts on the data from the highest layer to the lowest layer and vice versa if necessary. There is no need to reference an element of a layer from another for it to change, Google's library automatically takes care of it. This makes it much easier to change one of the layers downstream or to perform unit tests, i.e., a procedure allowing to individually test different parts of a program.[67]

5.7.11.2 The Data Binding library

The Data Binding library is the cornerstone of the MVVM on Android. Available for free, this library integrates a project almost like any other,[68] with the difference that it's also necessary to enable it in the build.gradle file[69] as follows[70]:

```
android {
    ....
    dataBinding {
        enabled = true
    }
}
```

Linking an item in the view to a variable of the ViewModel layer must be done in the XML:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <variable name="user" type="com.example.UserViewModel"/>
    </data>
```

[66] cf. 5.7.11.2The Data Binding library
[67] (Unit Testing, 2016)
[68] cf. 4.2 Gradle
[69] cf. 4.2 Gradle
[70] (Data Binding Guide | Android Developers, s.d.)

```
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <EditText android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@={user.mail}"/>

        <EditText android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@={user.password}"/>

    </LinearLayout>
</layout>
```

The following line describes the type of the ViewModel that is linked to the view:

```
<variable name="user" type="com.example.UserViewModel"/>
```

And that line is responsible for binding the ViewModel's properties to the desired view's attribute:

```
android:text="@={user.mail}"/>
```

Afterwards, it's necessary to go to the UserViewModel class to get to the representation of the data:

```
package com.andrew.fictional_login.viewmodel;

import android.databinding.BaseObservable;
import android.databinding.Bindable;

import com.andrew.fictional_login.BR;

public class UserViewModel extends BaseObservable {
    @Bindable
    private String mail;
    @Bindable
    private String password;

    public UserViewModel(String mail, String password) {
```

```
        this.mail = mail;
        this.password = password;
    }

    public String getMail() {
        return this.mail;
    }

    public String getPassword() {
        return this.password;
    }

    public void setMail(String mail) {
        this.mail = mail;
        notifyPropertyChanged(BR.mail);
    }

    public void setPassword(String password) {
        this.password = password;
        notifyPropertyChanged(BR.password);
    }
}
```

Although very similar to a conventional object class in Java, this class differs in the presence of two elements. The annotation "@Bindable" is used to generate the required entries in the BR class during compilation while "NotifyPropertyChanged()" is used to notify the View layer for it to apply the changes to the interface in real-time. The BR class is generated by the Data Binding library and regroup the identifiers of the elements annotated with a "@Bindable" contained in the classes belonging to the model layer. The system uses these references to run the data binding.[71]

This is what the BR class of the above example looks like:

```
package com.andrew.fictional_login;

public class BR {
    public static final int _all = 0;
    public static final int mail = 1;
    public static final int password = 2;
    public static final int user = 3;
}
```

It's then necessary to trigger the binding at runtime. This step has to be done in the business code of the views, specifically in the onCreate() method of the view.[72] It is also

[71] (CAMPBELL, 2015)
[72] cf.5.7.4.1 Activity lifecycle, 5.7.5 Fragment

necessary to report the binding when converting the XML code of the view into an object of type View usable through Java code:

```
private ActivityLoginBinding binding =
DataBindingUtil.setContentView(this, R.layout.activity_login);
binding.setUser(new UserViewModel("mailExample@example.com",
"passwordExample"));
```

5.8 Creation of a fictitious project

The main concepts required to be able to develop on Android now being established, it is therefore appropriate to focus on what makes the development of an app in practice. This is why this part is dedicated to the creation of a fictitious project, an introduction to get an overall but nevertheless representative idea of the ins and outs of the realization of such a project. The last elements specific to the code of an Android app are also discussed here, e.g., the project structure and its various resources.

The detailed explanation of these points is also needed to comfortably approach the key project of this work without having to come back on each concept along the way.

Figure 33 : fictional_login main activity

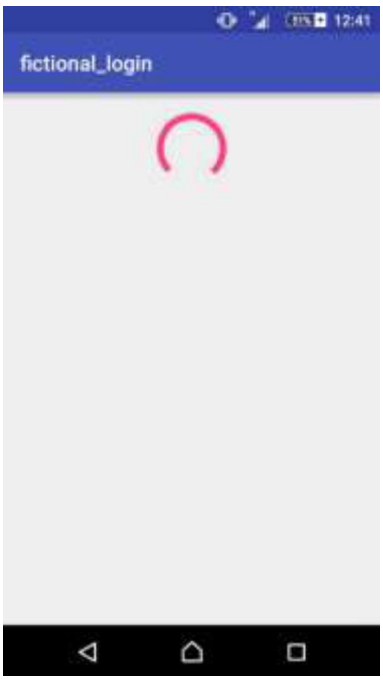

Figure 34 : fictional_login loading screen

5.8.1 Description of the fictitious app

Since most apps require the user to connect before it can be used, a wise choice is to create a project that simulates signing in. Thus, the example - name "fictional_login" - consists of an activity with two fields for inputting an email address and a password in addition to a button triggering the connection request. The request is represented by a false loading followed by the cease of the app.

5.8.2 Structure

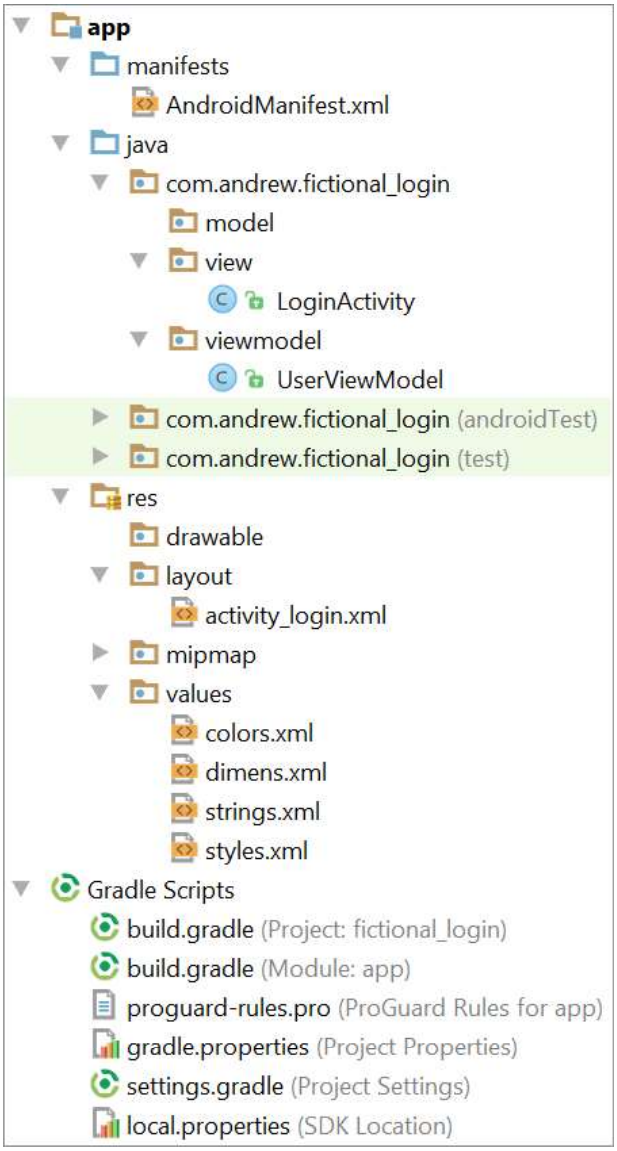

Figure 35 : an Android project structure

An Android project can be of significant size, it is therefore necessary to master its structure in order to navigate within it as effectively as possible. Also, a good structure ensures a better readability and an easier access to the different resources.

Android Studio[73] proposes to divide the project the following way (see Figure 35 : an Android):

- **manifests**

The "manifests" module contains the AndroidManifest.xml file,[74] a file describing the app, essential to all Android projects.

- **java**

The "java" module is devoted to the storage of every .java files. It's where the developer can place the code and better manage the disposal of the different packages – according, for example, to the MVVM design pattern[75] – in order to be able to navigate his classes.

[73] cf. 4.1 Android Studio
[74] cf. 5.8.3 AndroidManifest.xml
[75] cf. 5.7.11 MVVM and data binding

- **res/drawable**

This module is used to store the various visual resources (.jpg, .png, .gif, etc.) used by the app.

- **res/layout**

It contains different layout files, written in XML, used for constructing the views and for binding the ViewModel layer components to the interface when using the Data Binding library.[76] Here lies "activity_login.xml", i.e., the layout of the login page.

- **res/values**

This last module is dedicated to storing the different values that the app needs. Whether it's the hexadecimal color values to use for the visual theme of the application, the dimensions of the elements of the interface or the various texts used in the app, this is where it is advisable to declare them.

Also, these values make possible the management of several languages but this feature needs the developer to store the various string resources here and to access them according to the device's configured language.

5.8.3 AndroidManifest.xml

AndroidManifest.xml is a key file to any Android app. It's needed to identify the app and to establish certain of its properties.[77] Its primary function is to determine the app package. It also describes the different components of the app, i.e., the different services and activities[78] of the project. Among the information provided by this file can also be found a list of the different permissions that the user must grant the app in order to use all its features.

Here is the manifest of the project presented here:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.andrew.fictional_login"
    android:versionCode="1"
    android:versionName="1.0">
```

[76] cf. 5.7.11 MVVM and data binding
[77] (App Manifest | Android Developers, s.d.)
[78] cf. 5.7.4 Activity

```
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="23" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".view.LoginActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The first few lines determine the package name, here "com.andrew.fictional_login", and the version of the application, the name being the app identifier within the device:

```
package="com.andrew.fictional_login"
android:versionCode="1"
android:versionName="1.0">
```

The following line determines the permissions needed by the app. This one needing to sign the user in, an access to internet is needed:

```
<uses-permission android:name="android.permission.INTERNET" />
```

The "application" tag has attributes such as the name, the launch icon to use or the theme. Values starting with "@" are actually the constants identifiers stored in the "res" module.[79]

[79] cf. 5.8.2 Structure

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

Finally, still in the "application" tag is a list of the various activities of the project and their function. Here, the activity "LoginActivity" is, as the tag "intent-filter" specifies it, the first activity to be launched as well as the main activity of the application.

```
<activity
    android:name=".view.LoginActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5.8.4 Resources

If the Java code of an Android app is at the heart of the development, it is inseparable from the different resources used. This term actually means the different layouts and other values needed to run the program, such as the strings, the dimensions or the colors hexadecimal values needed for the style of the app. The few explanations that follow are devoted to the most frequently discussed resources.

5.8.4.1 strings.xml

Including the texts of the application, this file also allows running a multilingual app. By creating a directory "value-xx" within strings.xml - xx being the language code – and by adding the translated strings, the developer allows the app to display the proper text at launch.

The file "strings.xml" of fictional_login contains the title of the app and some indication messages or feedback messages aimed at the user:

```
<resources>
    <string name="app_name">fictional_login</string>

    <string name="prompt_email">Email</string>
    <string name="prompt_password">Password (optional)</string>
    <string name="action_sign_in">Sign in or register</string>
    <string name="action_sign_in_short">Sign in</string>
    <string name="error_invalid_email">This email address is
invalid</string>
    <string name="error_invalid_password">This password is too
short</string>
    <string name="error_incorrect_password">This password is
incorrect</string>
    <string name="error_field_required">This field is required</string>
</resources>
```

5.8.4.2 DIMENS.XML

This file contains the dimensions, in dp or sp,[80] of the different visual elements of the app such as the fonts used or the size of the internal margins of the designs.

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

5.8.4.3 STYLES.XML

It is customary to have to respect a graphic charter when creating a mobile app. Usually dictated by whom ordered it, this graphic charter is embodied during the development by the "styles.xml" file. Thus, here are defined the visual themes to use.

The example shown here uses the default theme provided by Android Studio[81]:

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

[80] cf. 5.6 Responsive design and material design
[81] cf. 4.1 Android Studio

5.8.4.4 COLORS.XML

In order to comply with a graphic charter and to facilitate the implementation of the themes to be used, it's necessary to declare the hexadecimal color values in the "colors.xml" file. This makes it easier to reference them later in the code.

For fictional_login, this resource is used to declare the colors used by the default theme of Android Studio, namely, two shades of blue and pink:

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

5.8.4.5 LAYOUTS

The layouts are the files in which are declared in XML virtually all graphical components. Similarly to the management of the strings when the app uses different languages, it is possible here to specify multiple layouts to choose based on the size of the screen that displays the app. In order to do so, different sub-folders are created with the name "layout-xxx", where xxx is the pixel density of the device. [82]

The layout of the example's login page is declared as follows:

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable name="user"
type="com.andrew.fictional_login.viewmodel.UserViewModel"/>
    </data>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context=".view.LoginActivity">

        <ProgressBar
            android:id="@+id/login_progress"
```

[82] cf. 5.6 Responsive design and material design

```
            style="?android:attr/progressBarStyleLarge"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="8dp"
            android:visibility="gone" />

        <ScrollView
            android:id="@+id/login_form"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <LinearLayout
                android:id="@+id/email_login_form"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <android.support.design.widget.TextInputLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content">

                    <EditText
                        android:id="@+id/email"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:hint="@string/prompt_email"
                        android:inputType="textEmailAddress"
                        android:maxLines="1"
                        android:singleLine="true"
                        android:text="@={user.mail}" />

                </android.support.design.widget.TextInputLayout>

                <android.support.design.widget.TextInputLayout
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content">

                    <EditText
                        android:id="@+id/password"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:hint="@string/prompt_password"
                        android:inputType="textPassword"
                        android:maxLines="1"
                        android:singleLine="true"
                        android:text="@={user.password}" />

                </android.support.design.widget.TextInputLayout>

                <Button
                    android:id="@+id/email_sign_in_button"
                    style="?android:textAppearanceSmall"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="16dp"
                    android:text="@string/action_sign_in_short"
                    android:textStyle="bold" />

            </LinearLayout>
        </ScrollView>
```

```
    </LinearLayout>
</layout>
```

Writing such a code is laborious, therefore it is worth remembering that Android Studio[83] has a tool for editing layouts operating on the principle of drag and drop to automatically generate the XML code.

If the "data" tag is dedicated to data binding,[84] others embody either views, i.e., different graphical components of Android, or containers, i.e., invisible elements extending the ViewGroup class and facilitating the configuration of these views. Among the different containers, these are the most regularly used[85]:

- **AbsoluteLayout**

This layout allows to specify the exact position of its components in terms of x/y coordinates.

- **FrameLayout**

FrameLayout is intended to devote a part of the screen to a specific item. The element in question is placed by default in the top left of the layout but changing its position can be done using the property "android:gravity" which is responsible for managing the alignment of an item.

- **GridLayout**

As its name suggests it, GridLayout facilitates the arrangement of its components in the form of a grid.

- **LinearLayout**

Surely one of the most useful layouts and therefore one of the most used, LinearLayout manages the arrangement of its components in a row or in a column. The attribute "android:orientation" is used to indicate the desired layout while "android: gravity" allows, similarly to FrameLayout, to indicate the alignment of the elements within it.

[83] cf. 4.1 Android Studio
[84] cf. 5.7.11.2 The Data Binding library
[85] (ViewGroup | Android Developers, s.d.)

- **RelativeLayout**

RelativeLayout, another one of the most practical layout, is used to specify how the elements organize themselves in relation to each other.

Among the remaining tags in the layout of fictional_login can be found "EditText", a field of inputting text, "Button", a button, "ProgressBar", the animated view indicating a loading, "ScrollView", which gives the ability to scroll the screen, and "TextInputLayout", an animated view related to "EditText" and specific to the material design.[86]

Each item must have at least two attributes, namely "android:layout_width", used to indicate the width, and "android:layout_height" for the height. If it is possible to specify a value in dp,[87] it is although appropriate to use the values "WRAP_CONTENT" or "MATCH_PARENT" for fixing the size according to the size of a child component or to the size of a parent component.

There are still a lot of kinds of views left, e.g., "TextView", used to display fixed text, or the "CheckBox" and all kinds of attributes. That's why developers are pleased to consult Android Developers, Google's reference site aimed at Android developers.[88]

5.8.5 R.JAVA

The R.java file is automatically generated by Android Studio[89] at compilation time. It actually contains the references to all of the "res" module elements and it is through this file that it is possible to access various resources from the source code of the app.

5.8.6 SOURCE FILES

Located in the "java" folder,[90] the source files relate to the implementation of the services, the activities and the methods of the app. Usually arranged by class, these files are of course those that require the most attention during the development of the app and those that are the most significant in its working.

The fictional_login project contains a single class, i.e., LoginActivity, the following being its implementation, partly generated by Android Studio - which facilitates the

[86] cf. 5.6 Responsive design and material design
[87] cf. 5.6 Responsive design and material design
[88] (Android Developers, s.d.)
[89] cf. 4.1 Android Studio
[90] cf. 5.8.2 Structure

creation of an activity dedicated to the identification - and adapted to be used here as an example:

```
package com.andrew.fictional_login.view;

import ...

public class LoginActivity extends AppCompatActivity {

    private UserLoginTask mAuthTask = null;

    private EditText mEmailView;
    private EditText mPasswordView;
    private View mProgressView;
    private View mLoginFormView;

    private ActivityLoginBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = DataBindingUtil.setContentView(this,
R.layout.activity_login);
        binding.setUser(new UserViewModel("mailExample@example.com",
"passwordExample"));

        mEmailView = binding.email;
        mPasswordView = binding.password;

        Button mEmailSignInButton = binding.emailSignInButton;
        mEmailSignInButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                attemptLogin();
            }
        });

        mLoginFormView = binding.loginForm;
        mProgressView = binding.loginProgress;
    }

    private void attemptLogin() {
        if (mAuthTask != null) {
            return;
        }


        mEmailView.setError(null);
        mPasswordView.setError(null);

        String email = binding.getUser().getMail();
        String password = binding.getUser().getPassword();

        boolean cancel = false;
        View focusView = null;
```

```
        if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {

mPasswordView.setError(getString(R.string.error_invalid_password));
            focusView = mPasswordView;
            cancel = true;
        }

        if (TextUtils.isEmpty(email)) {

mEmailView.setError(getString(R.string.error_field_required));
            focusView = mEmailView;
            cancel = true;
        } else if (!isEmailValid(email)) {
            mEmailView.setError(getString(R.string.error_invalid_email));
            focusView = mEmailView;
            cancel = true;
        }

        if (cancel) {
            focusView.requestFocus();
        } else {
            showProgress(true);
            mAuthTask = new UserLoginTask(email, password);
            mAuthTask.execute((Void) null);
        }
    }

    private boolean isEmailValid(String email) {
        return email.contains("@");
    }

    private boolean isPasswordValid(String password) {
        return password.length() > 4;
    }

    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
    private void showProgress(final boolean show) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
            int shortAnimTime =
getResources().getInteger(android.R.integer.config_shortAnimTime);

            mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);
            mLoginFormView.animate().setDuration(shortAnimTime).alpha(
                    show ? 0 : 1).setListener(new
AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);
                }
            });

            mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
            mProgressView.animate().setDuration(shortAnimTime).alpha(
                    show ? 1 : 0).setListener(new
AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mProgressView.setVisibility(show ? View.VISIBLE :
View.GONE);
```

```
                }
            });
        } else {

            mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
            mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);
        }
    }

    public class UserLoginTask extends AsyncTask<Void, Void, Boolean> {

        private final String mEmail;
        private final String mPassword;

        UserLoginTask(String email, String password) {
            mEmail = email;
            mPassword = password;
        }

        @Override
        protected Boolean doInBackground(Void... params) {
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                return false;
            }
            return true;
        }

        @Override
        protected void onPostExecute(final Boolean success) {
            mAuthTask = null;
            showProgress(false);

            if (success) {
                finish();
            } else {

mPasswordView.setError(getString(R.string.error_incorrect_password));
                mPasswordView.requestFocus();
            }
        }

        @Override
        protected void onCancelled() {
            mAuthTask = null;
            showProgress(false);
        }
    }
}
```

LoginActivity extends AppCompatActivity, a class of the Support Library[91] extending itself the Activity class.[92] The onCreate()[93] method is redefined to execute the

[91] cf. 5.7.10 Android support library
[92] cf. 5.7.4 Activity
[93] cf. 5.7.4.1 Activity lifecycle

binding and then to initialize the components and the variables. It is in this method that the content of the view is described.

The button (see Figure 33 : fictional_login main activity) is declared with an event listener so that clicking it executes the attemptLogin() method that will retrieve the email address and the password inputted by the user using data binding[94], checks whether they are valid using the isPasswordValid() method and the isEmailValid() method and launches an asynchronous task simulating a connection attempt and displaying a loading animation.

The isEmailValid() method checks if the email address entered by the user contains an "@" and isPasswordValid() verifies that the password has a length of more than four characters.

The asynchronous task pauses the app for ten thousand milliseconds, the time for a loading animation (see Figure 34 : fictional_login loading screen) to occur. At the end of these ten thousand milliseconds, the application stops.

If the email address and the password do not match - the way it has previously been declared in onCreate() in order to simulate the remote database - is also taken into account but this application being fictitious, it actually has no effect on the interface.

[94] cf. 5.7.11.2 The Data Binding library

6 Presentation of the project

The first two weeks of this internship were dedicated to a training exercise, namely the making of a functional mobile app. The thirteen following weeks were devoted to the development of another app, Ethias Prevention. This order from Ethias is intended to assist its field insurers. Thus, an agent can visit a client and, with no need for any book or pen, can write damage description reports, risk screening reports, status reports, incident reports or different audit types directly on the app which will then take care of the centralization.

The application is as follows: the user arrives at a login page at launch, provided with a menu for choosing the language and a solution in case he forgot his password. This connection is based on data already available at Ethias and leads to the homepage. On this second screen is a list of current and pending reports and a visual representation - a chart and a map of Belgium - of this list. From this page, the agent can directly access the creation of a new report and two tabs: "report" and "historic". The "report" page consists of a form that can be filled with the name and the reference number of the report, the customer's name, the visiting date, the type of mission, the location - accessible from a Google Maps widget, the list of participants and the list of recipients. These reports can be enriched by sections whose agent decides the nature of and to which he can add pictograms, legislations, a level of risk and priority and videos, photos or audio files. Finally, it can also record his observations and recommendations before submitting the report. The last page, history, consists of a list of finalized reports that can be printed or sent by email.

The mission is therefore to continue or finish the project, adapting it to the new Android technologies, namely the Data Binding library made recently available by Google and the related design pattern, the MVVM, i.e., the Model View ViewModel.[95]

6.1 Analysis

The features of the app are decided. However, such work requires a high attention in order to not dive into an endless development filled with pitfalls. This is why this part is dedicated to the analysis of the desired functionalities of the app. Among the items detailed below can be found the different features of the app in the form of use cases, the status evolution of a report from its creation to its completion, the browsing possibilities within the app and the non-functional requirements of the latter, i.e., its features and business objects centered around its looks and ergonomics.

[95] cf. 5.7.11 MVVM and data binding

6.1.1 ACTIVITY DIAGRAM

An activity diagram is a concise representation of the different activities and their progress as well as the evolution of the business object, i.e., the report. It also facilitates the analysis of the different use cases.

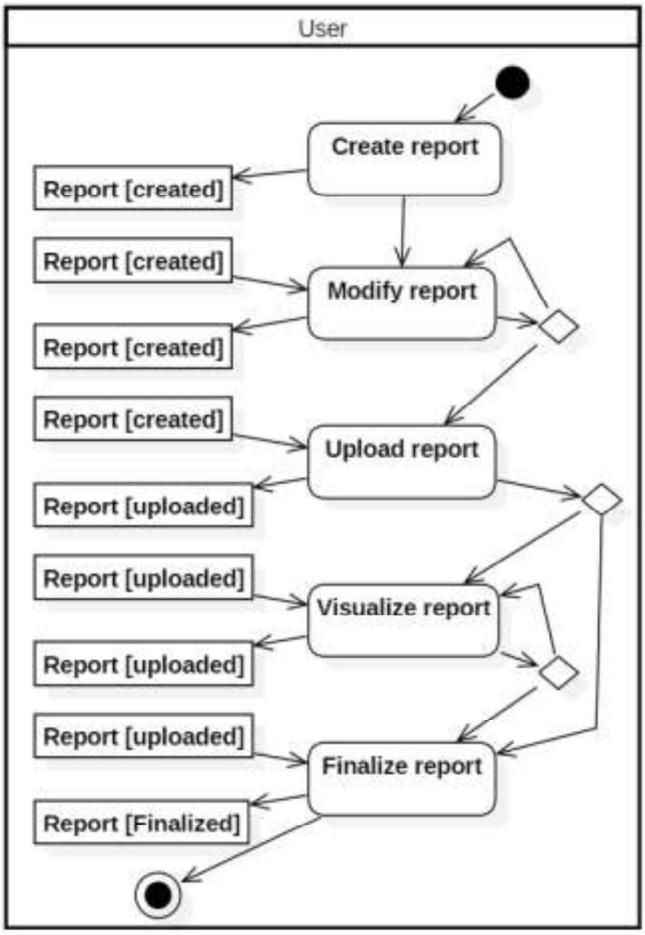

Figure 36 : activity diagram

6.1.2 STATE DIAGRAM

The state diagram shown here addresses the evolution of the report state based on the progress of its completion.

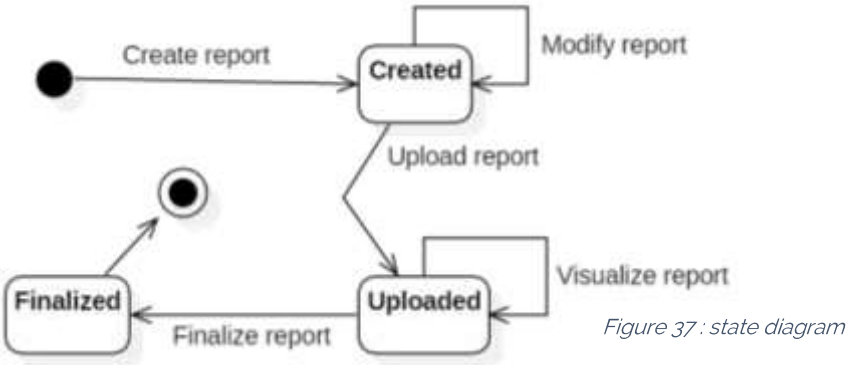

Figure 37 : state diagram

6.1.3 USE CASE DIAGRAM

This section specifies the different use cases extracted from the activity diagram. The five steps leading to the making of a report from creation to completion, are presented here in details.

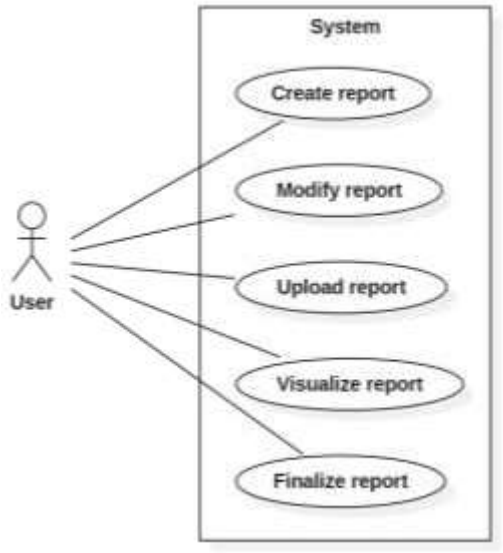

Figure 38 : use case diagram

6.1.3.1 CASE 1 : CREATE REPORT

The use case begins when the user wants to create a new report. He therefore makes it to the page for creating a report, completes the fields[96] and then saves it. At the end of this case, the status of the report is "created" or "in progress".

6.1.3.2 CASE 2 : MODIFY REPORT

Accessible from the homepage, the list of current reports - and therefore the ones saved only on the device for now - allows to modify the reports. The user has to click on the edit button of the item to modify in the list to be taken to the reports creation screen with the fields already completed according to the last backup. He can change the items he wants and save the report. This use case can be repeated at will and does not change the status of the report.

6.1.3.3 CASE 3 : UPLOAD REPORT

Once the report satisfyingly filled, the user can upload it to the database so that it is remotely accessible from other devices. In order to do so, the user simply has to click the button to upload the report in the list of current reports. Once uploaded, the report

[96] cf. 6.2.4 Reports creation screen

can only be changed from the backend, i.e., the web tool associated with the app used to access the database. The status of the report then goes to "standby" which is reflected by the interface.

6.1.3.4 Case 4 : Visualize report

Now in the pending reports list, still accessible from the home page, the report can be viewed by the user. In order to do so, he has to click on the report in the list. This will take him to the creation screen but this time devoid of the ability to edit the fields. Visualization is also possible from the reports history[97] and can be repeated at will.

6.1.3.5 Case 5 : Finalize report

The report can also be finalized from the list of pending items. The user simply has to click the appropriate button to change the status of the report. This has the effect of making it move into the reports history and of changing its status to "finalized".

[97] cf. 6.2.5 Reports history

6.1.4 CLASS DIAGRAM

The following class diagram presents the different types of entities involved in the application as well as their arguments.

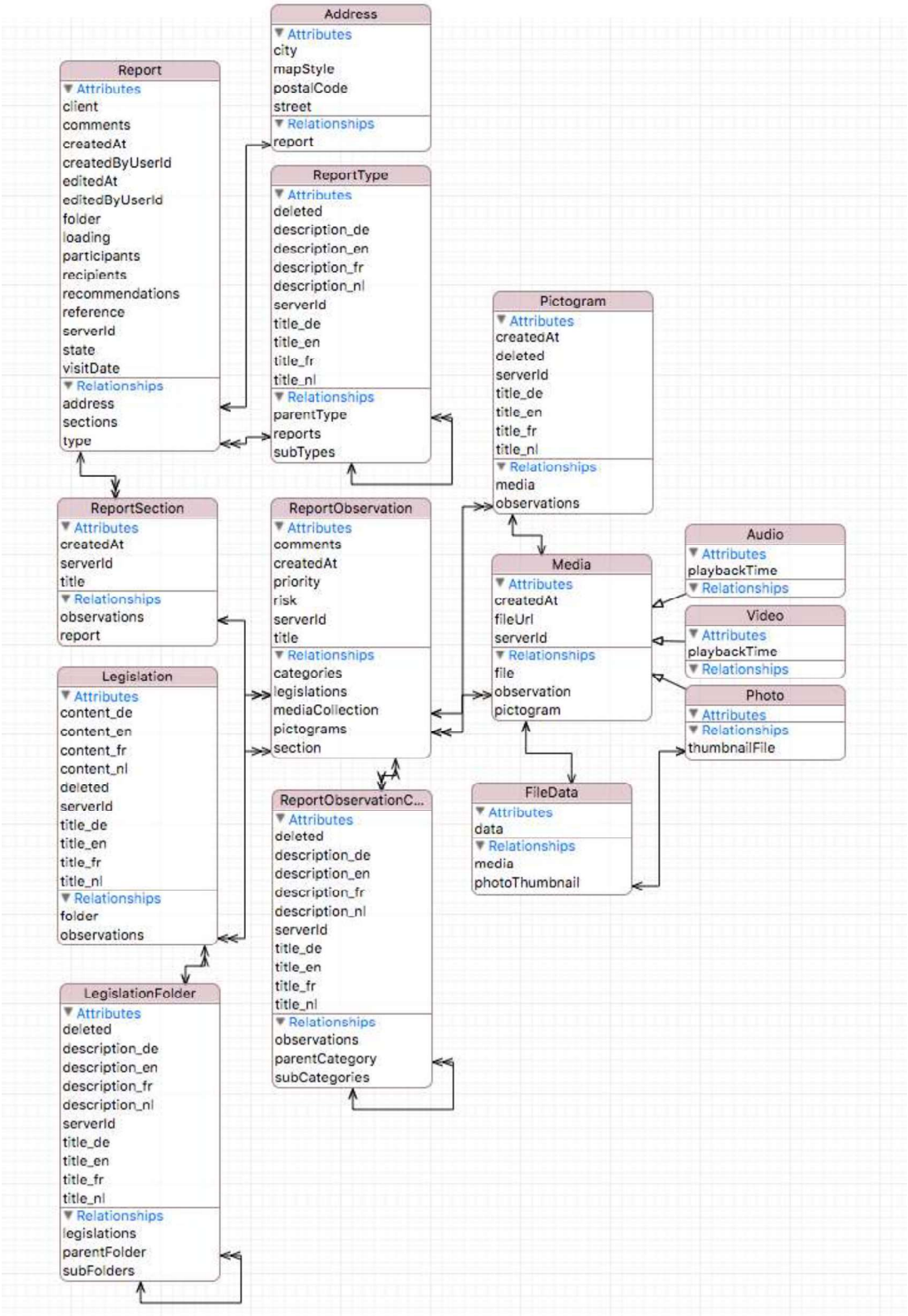

Figure 39 : class diagram, generated using Core Data for the iOS version of Ethias Prevention

6.1.5 NAVIGATION DIAGRAM

The following navigation diagram is used to represent the working of the app by illustrating the different navigation possibilities that are offered to the user.

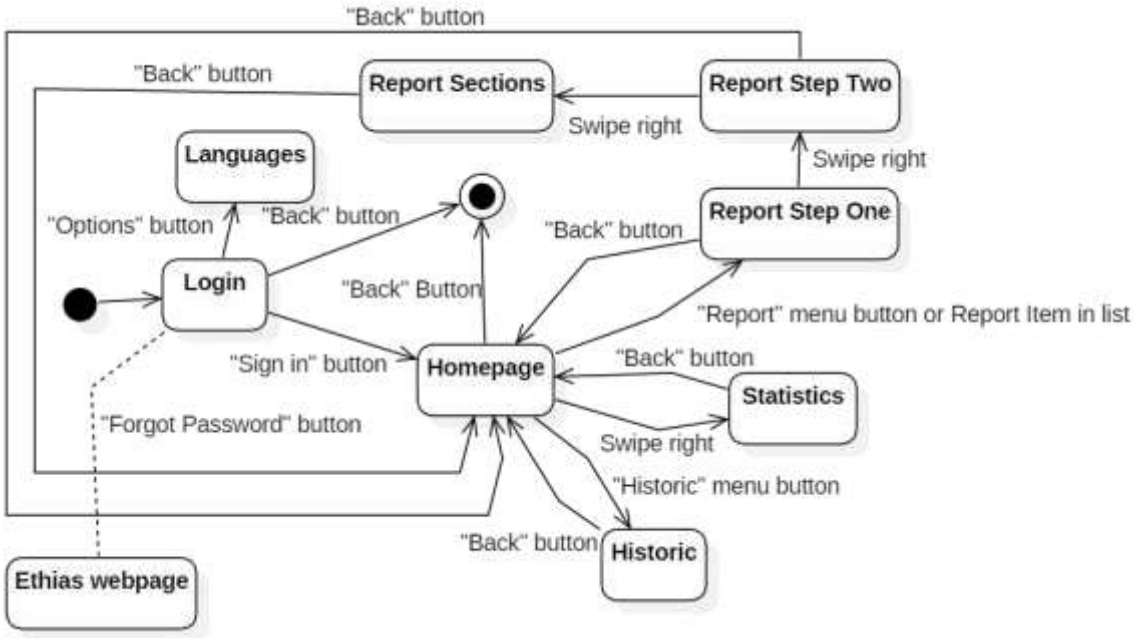

Figure 40 : Navigation diagram

6.1.6 NON-FUNCTIONAL REQUIREMENTS

Ethias Prevention is a mobile app. Accordingly, it must be adaptable to multiple screen sizes and its use must be ergonomic and intuitive.[98] This also implies the fluidity, the clarity, the effectiveness and the practicality of the program.

It must also be usable in case of malfunction of the internet connection, a copy of the database must then be stored on the device.[99] Developed according to the MVVM[100] design pattern, the application must be testable and easily modifiable.

Finally, it must be easily obtainable, i.e., downloadable via Google Play.[101]

6.2 GUIDED TOUR OF ETHIAS PREVENTION

The analysis of the project now being carried out, it is therefore appropriate to focus on the end product. This section is dedicated to the guided tour of the app,

[98] cf. 5.6 Responsive design and material design
[99] cf. 6.3.2 Realm to persist the data
[100] cf. 5.7.11 MVVM and data binding
[101] cf. 5.3 Google Play

enriched with screenshots, explanations on the development and comments justifying or criticizing the absence or presence of certain features.

6.2.1 LOGIN PAGE

Figure 41 : screenshot of the login page

The login screen is the gateway to the application. Each user must login before he can use the different features of Ethias Prevention. The login page is already functional without any problems and can identify a user to whom Ethias would have provided an adequate email address and a correct password but it is not yet complete.

Indeed, the "Forgot your password?" feature has no effect so far, its implementation requiring the intervention of Ethias. The "tool" button in the upper right that gives access to the language selection is meanwhile already effective. Although, resources in various languages have not yet been translated.

Also, such a page should be able to memorize the previous values entered by the user and provide the email address linked to the phone when he clicks the "Email" field, which is not the case. Ideally, this page should not be displayed each time the app is started because the user should not have to reconnect every time but this feature remains to be implemented.

Finally, the "Login" button launches an identification request via a web service[102] and then loads the information in the database[103] to allow the user to work with a limited internet connection. After these operations, the app homepage[104] appears.

[102] cf. 6.3.1 Retrofit
[103] cf. 6.3.2 Realm
[104] cf. 6.2.2 Homepage

6.2.2 HOMEPAGE

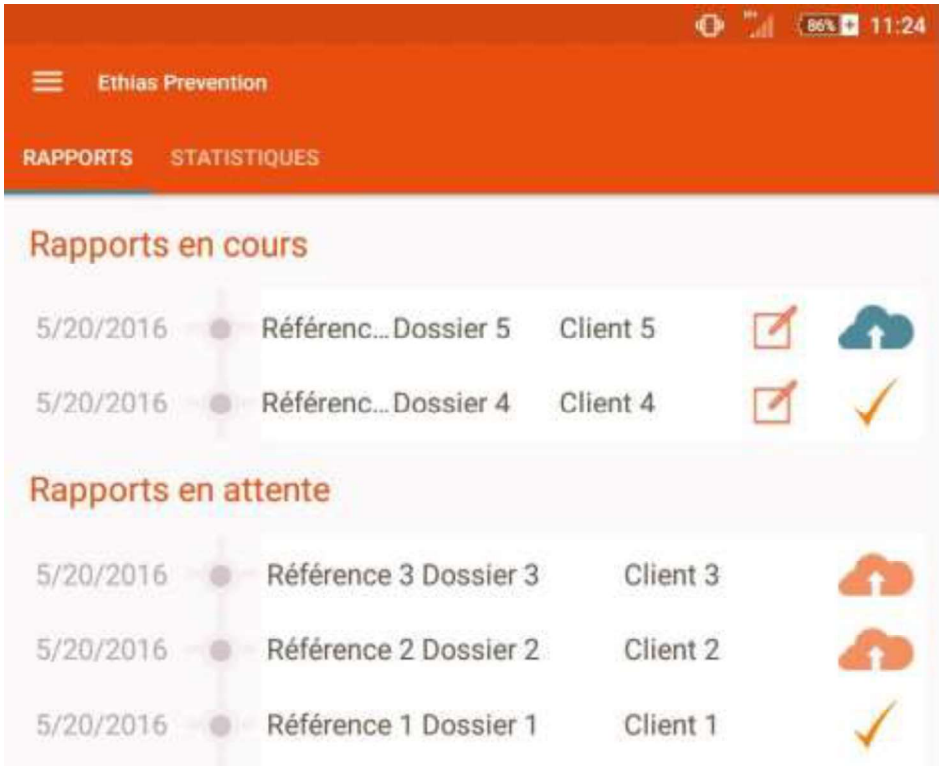

Figure 42 : screenshot of the homepage

The homepage, here displayed in its horizontal arrangement in order to see a maximum of information, aims to remind the current reports and those waiting to the user. These reports have been loaded from the local database and are displayed in a list of type "RecyclerView". Such a list is actually a layout[105] provided by Android that allows developers to arrange the elements as they wish using an adapter.[106] This list is designed to optimize memory usage by loading only what is displayed on the screen and by recycling items into new ones once they disappear from the user's view.

The strings "Rapports en cours" and "Rapports en attente" are the headers of the list but they follow the list for the moment when it scrolls because of a user action. Ideally, these should remain on the screen while the user would scroll the list below them. However, this feature, called "Sticky Header", remains to be implemented.

The icon in the shape of a note and a pencil is used, as its appearance suggests, to modify the current report. Clicking this button has the effect of opening the report creation page[107] filled beforehand with the different values of the selected report. From the implementation point of view, when the user clicks on this icon, it actually creates

[105] cf. 5.8.4.5 Layouts
[106] cf. 5.7.9 Adapter
[107] cf. 6.2.4 Reports creation screen

an intent[108] in which these values are stored - retrieved from the local database[109] - and the name of the activity to start, namely that of the report creation. It therefore engages the end of the current activity and eventually redirects the user to the appropriate screen.

The button to the right of it, in the shape of a blue cloud, is used to upload the report on the database through a web service.[110] In fact, this action only changes one argument - the status of the report - and the success of the operation is indicated to the user with a confirmation icon in the shape of a "V". After the upload, once the page reloaded, the report will not be present in the list of current reports anymore, but in the pending reports one.

Finally, the orange cloud-shaped button is used to change the status of the report, again using a web service, so it is specified as finalized. Therefore, once the page reloaded, the report disappears from the list of the current reports and goes to the reports history.[111]

The "Statistics" tab in the top of the screen is supposed to display a graphical representation of various statistics about the reports. This should be in the form of a Belgium map that displays circles in some places whose size depends on the number of reports made at these places as well as in the form of a graph of the number of reports created by date. However, this part, more technical, remains to be implemented.

[108] cf. 5.7.7 Intent
[109] cf. 6.3.2 Realm
[110] cf. 6.3.1 Retrofit
[111] cf. 6.2.5 Reports history

6.2.3 Menu

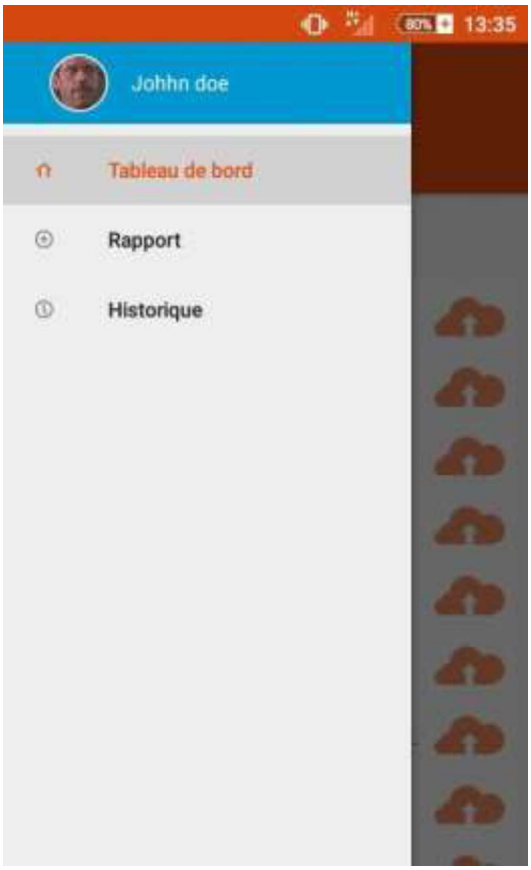

Figure 43 : screenshot of the menu

The menu, which is in the form of a "Navigation Drawer", is accessible from the home screen[112], the screen for creating reports[113] and the one displaying the reports history[114] by a swipe[115] from the far left of the screen to the right of it or by clicking on the "Hamburger" button, an icon consisting of three horizontal lines referenced in Google's guide on material design.[116]

This sliding menu is implemented using XML[117] with a "DrawerLayout" tag that includes the content of the page and in which a tag can be implemented, the "NavigationView" one representing a standard navigation menu. The latter can be enriched with a "menu" tag that is used to arrange the elements as desired.

To illustrate this explanation, here is the layout related to the "Navigation Drawer":

```
<android.support.v4.widget.DrawerLayout
    android:id="@+id/activity_base_drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include
        android:id="@+id/activity_base_nav_content_fragment"
        layout="@layout/activity_base_nav_content_fragment" />

    <android.support.design.widget.NavigationView
        android:id="@+id/activity_base_navigation_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/menu_toolbar"
        app:menu="@menu/menu_navigation_view" />
```

[112] cf. 6.2.2 Homepage
[113] cf. 6.2.4 Reports creation screen
[114] cf. 6.2.5 Reports history
[115] cf. 5.7.8 Gestures
[116] cf. 5.6 Responsive design and material design
[117] cf. 5.8.4.5 Layouts

And here's the resource referenced in this layout, used to arrange the menu:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/menu_navigation_view_dashboard"
            android:icon="@drawable/ep_menu_dashboard"
            android:title="@string/menu_dashboard" />
        <item
            android:id="@+id/menu_navigation_view_report"
            android:icon="@drawable/ep_menu_rapport"
            android:title="@string/menu_rapport" />

        <item
            android:id="@+id/menu_navigation_view_historic"
            android:icon="@drawable/ep_menu_history"
            android:title="@string/menu_historique" />
    </group>
</menu>
```

The aforementioned tags and the three menu items are indeed present.

When this is written, it is necessary to indicate in the source code[118] of the class that it implements "OnNavigationItemSelectedListener" and to redefine the method "onNavigationItemSelected()" to get a reference to the item selected by the user. This reference can then be used to load the specified page. In most activities[119] of Ethias Prevention, the listener is implemented as follows, "startActivityFromNavigationView()" being a method responsible for launching the activity corresponding to the received reference:

```
@Override
public boolean onNavigationItemSelected(MenuItem menuItem) {
    if (!super.onNavigationItemSelected(menuItem)) {
        NavigationController.startActivityFromNavigationView(this,
menuItem.getItemId());
        return true;
    }
    return false;
}
```

[118] cf. 5.8.6 Source files
[119] cf. 5.7.4 Activity

6.2.4 Reports creation screen

The reports creation screens are at the heart of Ethias Prevention since the activities[120] related to them are the main reason why Ethias commissioned this application. This creation of reports is made of three steps and therefore takes place on at least three screens. Once these three pages - or more - visited, the user can save the report in the local database[121] by clicking the confirmation button shaped as a "V" located at the top right of the screen. Saving has the effect of changing the report status to "created", which can be likened to "ongoing" since it is in a list under that name[122] that the report will be stored once created.

It is worth noting that the user can also edit a current report. In this case, it is this creation page that is displayed with the fields already filled with the information present in the local database. If he wants to view a pending or a finalized report, it is once again the creation screen that appears but without the ability to edit the fields.

[120] cf. 6.1.1 Activity diagram
[121] cf. 6.3.2 Realm
[122] cf. 6.2.2

6.2.4.1 STEP ONE, GENERAL INFORMATION

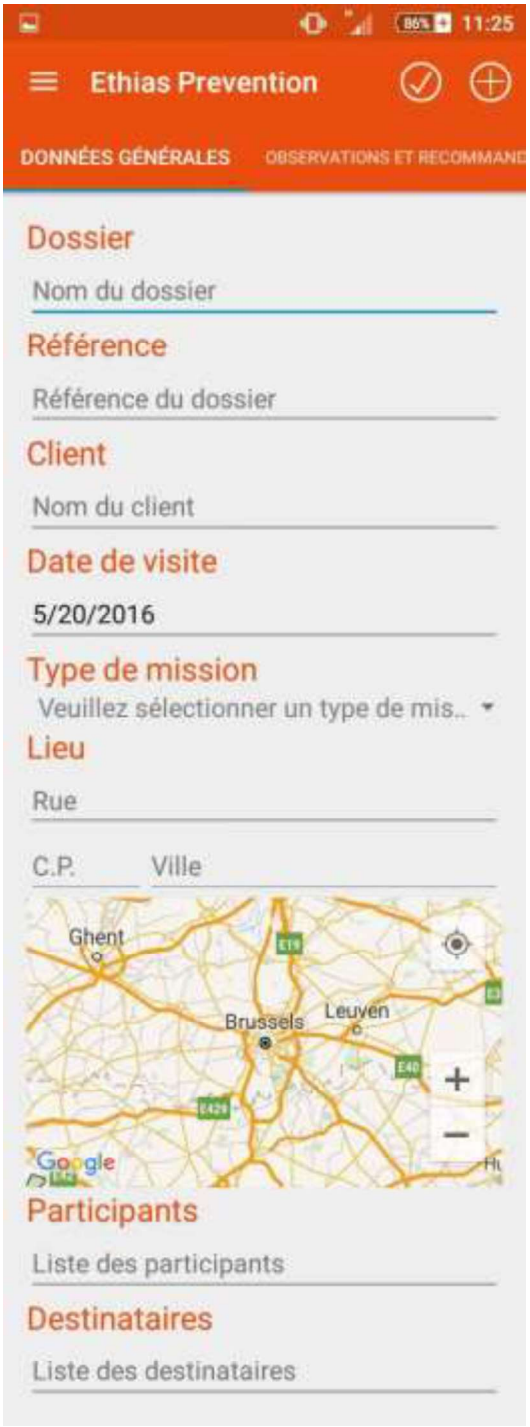

Figure 44 : screenshot of the "General information" screen

The general information page is used for the user to enter the name of the file and its reference, the customer's name, the visiting date, the address where the visit takes place, the type of mission as well as the list of participants and persons to whom the report is intended.

If most of the information is inputted using conventional fields in which one can only insert text, there is some information that the system can help to complete.

Clicking on the date field, already displaying the current date by default, opens a dialog helping the user to choose the date. This dialog is called "DatePickerDialog" and consists of a screen provided by Android which drops down the days, the months and the years that are possible to select. This screen is easily integrable since it's provided by Android Studio and it just needs to be called when the user interacts with the relevant field for him to be able to use it.

The mission type is selected in a "Spinner", a scrollable list layout[123] provided by Android Studio[124] that simply links[125] to information of the local database[126] using an adapter.[127]

Finally, it is possible to manually insert the address but the user can also let the Google Maps widget do the job. By doing a long click[128] on a location of the map - which has the effect of displaying a marker - or by clicking the geolocation button located on the top right

[123] cf. 5.8.4.5 Layouts
[124] cf. 4.1 Android Studio
[125] cf. 5.7.11.2 The Data Binding library
[126] cf. 6.3.2 Realm
[127] cf. 5.7.9 Adapter
[128] cf. 5.7.8 Gestures

of the latter, the widget will automatically fill the street field, the postcode field and the city field with the values specific to the selected location.

Although a common feature in Android applications, the integration of a Google Maps widget is not simple. The first stage of the implementation, once the view used for the map is created, is to get a Google Maps API key.[129] To do this, it's necessary to go to the dedicated page of the Google Developers website[130] and to follow the steps that are described there in order to record the project in the Google Developer Console and to receive the key. These steps are summarized as follows:

- **Getting the information about the app certificate**

It's first necessary to get the information on the app certificate - a SHA-1 key, SHA-1 being a method of cryptographic hash - generated by the Android SDK[131] that developers can get by entering a line in the Windows console. In the debug version of the application, i.e., the development version, the line in question is as follows:

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias
androiddebugkey -storepass android -keypass android
```

In the case of the release version, i.e., the version available to users, the following line needs to be typed:

```
keytool -list -v -keystore your_keystore_name -alias your_alias_name
```

Where "your_keystore_name" is the path of the keystore file obtained when publishing the app on Google Play.[132]

[129] (Premiers pas | Google Maps Android API | Google Developers, 2016)
[130] (Signature et clés d'API | Google Maps Android API | Google Developpers, 2016)
[131] cf. 5.4 Android Software Development Kit
[132] cf. 5.3 Google Play

- **Creating an API project in the Google Developers Console**

In order to create an API project, it's necessary to go to the Google Developers Console,[133] to create a new project in the name of the app and finally to activate the API "Google Maps Android API".

- **Getting the Android API key**

In that same developer console, to get the API key required for the Google Maps widget to work, it is first necessary to generate it. In order to do so, the console invites developers to enter the app certificate key previously obtained as well as the name of its package. Then, an API key is created.

- **Adding the API key to the app**

Finally, if all went accordingly, it is now necessary to save this key in the AndroidManifest.xml[134] file of the application by using the following lines:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY"/>
```

That's it regarding the part about the API key. The implementation of the map, meanwhile, is done as follows inside the layout[135] file of the relevant activity[136]:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/map"
tools:context=".MapsActivity"
android:name="com.google.android.gms.maps.SupportMapFragment" />
```

And like this in the class of the latter:

[133] (Bibliothèque d'API, s.d.)
[134] cf. 5.8.3 AndroidManifest.xml
[135] cf. 5.8.4.5 Layouts
[136] cf. 5.7.4 Activity

```
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap map) {
        // Add a marker in Sydney, Australia, and move the camera.
        LatLng sydney = new LatLng(-34, 151);
        map.addMarker(new MarkerOptions().position(sydney).title("Marker
in Sydney"));
        map.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

6.2.4.2 STEP TWO, OBSERVATIONS AND RECOMMENDATIONS

Figure 45 : screenshot of the "Observations and recommendations" screen

The observations and recommendations page, which could not be simpler, only consists of two text insertion fields dedicated to the various comments and recommendations that the employee might wish to add to the report.

6.2.4.3 STEP THREE, SECTIONS

Figure 46 : screenshot of a new section

An Ethias employee can add as many sections as he desires to a report by using the "+" shaped button located in the upper right corner of the screen. In doing so, they're added as a new tab in a "TabLayout", i.e., a layout for navigating from page to page by swiping[137] or by selecting the title of the tab to display, using an adapter.[138] They can correspond to a particular point of care or to a defective and dangerous machine.

These sections consist of a title and an undefined number of observations that may correspond to parts of these machines. Once again, the agents can add as many observations as they like using a "FloatingActionButton", i.e., a floating button described in Google's guide on material design[139], here located at the bottom right of the screen.

These observations consist of a subject, a selection of types of observations, a description and an estimation of the level of risk and priority. It should normally be possible to attach pictograms describing the risk as well as relevant laws and various media such as photos, videos or audio recordings recorded directly from the smartphone but even if the fields of some of these items is provided, the implementation of these functionalities remains to be completed.

The comments are dynamically added to a list of type "RecyclerView" - already addressed during the presentation of the homepage.[140] If their subject and description are added via a text insertion field, the types - just as it should be the case for pictograms and laws - are added using a "CustomSpinner", i.e., a "Spinner" modified so

[137] cf. 5.7.8 Gestures
[138] cf. 5.7.9 Adapter
[139] cf. 5.6 Responsive design and material design
[140] cf. 6.2.2 Homepage

that the selection of multiple items is possible. This "CustomSpinner" is linked[141] to the information available in the database using an adapter.

Finally, the level of risk and priority is chosen by the user through a "ProgressBar" - or a loading bar - customized using a library named "philjay.valuebar". This library provides colorful and responsive selection bars. Touching a location of the bar makes it grow or shrink according to the touched location. The bar also changes color depending on the choices of the developers. Ethias Prevention's ones have three levels that are colored from green to red through orange.

6.2.5 REPORTS HISTORY

Figure 47 : screenshot of the reports history

Ethias Prevention's last page is the reports history. Consisting of a "RecyclerView" list, discussed in the section about the homepage,[142] it shows the reports of which the state is "finalized". It is possible, by clicking on the icon of each report, to print or to send by email a PDF version of the report.[143]

The generation of a report .pdf file is taken care of by the backend. So, clicking on the "print" or "send by email" icon will first execute a web service[144] in order to download the corresponding PDF. After that, if the user wants to send the report by email, a dialog appears asking the latter to enter the recipient's address before an intent[145] is created using the .pdf file information, the recipient's address and a request to execute an already installed mail management app. The user can then choose the app to run in order from it to display the mail sending page with the recipient's

[141] cf. 5.7.11.2The Data Binding library
[142] cf. 6.2.2 Homepage
[143] cf. 11.1 A PDF report
[144] cf. 6.3.1 Retrofit
[145] cf. 5.7.7 Intent

address field and the mail title field already filled and with the PDF version of the report attached.

On the other hand, printing the report directly creates an intent containing the .pdf file and asking for an app devoted to the printing of documents to be executed.

6.3 Libraries used

Reinventing the wheel is the last thing developers want to achieve. This is why several libraries were used during the development of Ethias Prevention. While most of them just allow to avoid having to type uninteresting code or to create a graphic element more easily, there is although two libraries that need to be explained given their importance and popularity among Android apps.

6.3.1 Retrofit to handle the web services

The web services of an app are the services[146] dedicated to process data exposed on the Internet.[147] Typically, they return data in JSON - a generic textual data format easily readable by humans and understandable by machines[148] - but sometimes they are presented in XML or in other data formats.

While it is often laborious to implement such a service since they need a lot of boilerplate code, there still are alternatives greatly facilitating the task like programming libraries. These are intended to abstract much of the code necessary for the web services to work so that developers only have to complete the implementation with items specific to the app they're developing.

For Ethias Prevention, the Retrofit library[149] is the one that is being used. It transforms the HTTP API in a very convenient Java[150] interface. The API comes with Android Studio[151] and is used to create a client responsible for managing HTTP requests, i.e., requests that allow developers to obtain the desired data. Therefore, the interface of Ethias Prevention used to get a list of reports to display on the screen is implemented as follows:

[146] cf. 5.7.6 Service
[147] (Acesyde, 2012)
[148] (JSON, s.d.)
[149] (Retrofit, s.d.)
[150] cf. 5.7.1 Java
[151] cf. 4.1 Android Studio

```
package be.ethias.ethias_prevention_android_v2.model.retrofit;

import ...

public interface EthiasService {

    @GET("/api/reports")
    Call<List<Report>> getReports(@Header("Authorization") String token);

}
```

The getReports() method takes a token as argument, ensuring that the user has the permission to get the list of reports, and returns a list of reports already converted from the JSON format to Java objects through features native to Retrofit. The annotation "@GET" refers to the "GET" HTTP request and the "/api/reports" element is actually the rest of the URL to which the request is addressed. It is added here to the base URL "http://ethias-test.djm.eu" declared in the service class.

In the case of Ethias Prevention, and more specifically, of the request for a list of reports, the service class is ReportsService. It implements IReportsService, an interface used in order to decouple the service from the rest of the code and Callback<List<Report>>, a callback of the request, provided by Retrofit and used to access the received data:

```
package be.ethias.ethias_prevention_android_v2.model.services;

import ...

public class ReportsService implements IReportsService,
Callback<List<Report>> {

    @Inject
    public OkHttpClient okHttpClient;

    @Inject
    public Gson gson;

    private ReportsCallback reportsCallback;

    private Retrofit retrofit;

    public void getReports(@Nullable final ReportsCallback
reportsCallback, Context context) {

        EthiasApplication.getDefaultComponent(context).inject(this);

        this.reportsCallback = reportsCallback;

        retrofit = new Retrofit.Builder()
```

```
                .baseUrl("http://ethias-test.djm.eu")
                .client(okHttpClient)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        EthiasService ethiasService =
retrofit.create(EthiasService.class);
        Call<List<Report>> call =
ethiasService.getReports(EthiasApplication.getAppPreference().getAuthoriz
ationPrefValue().getValue());
        call.enqueue(this);
    }

    @Override
    public void onResponse(Call<List<Report>> call,
Response<List<Report>> response) {
        if (response.isSuccessful()) {
            reportsCallback.onReportsCallBack(response.body());
            return;
        }

        Error myError;
        try {
            myError = (Error) retrofit.responseBodyConverter(
                    Error.class, Error.class.getAnnotations())
                    .convert(response.errorBody());
            Log.d("onResponse ", "onResponse: " + myError.toString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

The body of the getReport() method involves first injecting the components such as the Gson object, useful to convert data received in JSON format to Java objects, and such as the HTTP client and then executing the request using Retrofit. The token ensuring user permissions is retrieved from the app's preferences, where it was previously saved using another web service, i.e., the identification one.

Finally, the onResponse() method is called when the query ends. If it is successful and the data is received, then a call to onReportsCallback() of the ReportsCallback interface allows the app to retrieve the data and to do what the developer dictated in the relevant class implementing onReportsCallback(). If it fails, the response is therefore converted to an Error object to which it is easy to access to figure out the problem.

Among the most convenient features of Retrofit, there are the asynchronous query management, the management of arguments, headers and commands – e.g. the sorting - in the accessed URL and the presence of different converters of data formats.

6.3.2 REALM TO PERSIST THE DATA

Ensuring the proper functioning of a connected app in all conditions can be tricky. To do this, it's necessary to persist the data on the device, to work on them when necessary, and finally to update the remote database only when the status of the Internet connection allows it. If remotely accessing and processing data belongs to the field of Retrofit,[152] their local management rather concerns Realm.[153]

Available in several versions like in Swift and in Objective-C - two programming languages from the Apple universe - and especially in Java,[154] Realm is a programming library that allows to easily and effectively persist data from the model layer[155] of an app in the memory of the device that is being used.

Its operating is very simple as shown in the example below:

```
@Override
public void onReportsCallBack(List<Report> reports) {
    RealmConfiguration realmConfig = EthiasApplication.getRealmConfig();
    Realm realm = Realm.getInstance(realmConfig);
    realm.beginTransaction();
    realm.copyToRealm(reports);
    realm.commitTransaction();
    startDashboardActivity();
}
```

The onReportsCallback() method is triggered at the reception of the reports from a remote database.[156] Realm allows to simply persist these reports by referencing an instance of the local database created at the app's launch and by copying them to it, the only condition being that the Report class model layer must extend the RealmObject class provided by the library.

Therefore, accessing a Report object in the local database can be done the following way:

```
RealmConfiguration realmConfig = EthiasApplication.getRealmConfig();
Realm realm = Realm.getInstance(realmConfig);
Report realmReport = realm.where(Report.class).equalTo("Id",
intent.getStringExtra(Consts.REPORT_ID_KEY)).findFirst();
```

[152] cf. 6.3.1 Retrofit
[153] (Java Docs - Realm is a mobile database: a replacement for SQLite & Core Data, 2016)
[154] cf. 5.7.1 Java
[155] cf. 5.7.11.1 Presentation of the MVVM
[156] cf. 6.3.1 Retrofit

Here, a reference to the report whose identifier is equal to the one recovered in the intent[157] is obtained and stored in the variable realmReport. To modify the obtained report, the developer does as follows:

```
realm.beginTransaction();
realmReport.setState(Consts.REPORT_STATE_FINALIZED);
realm.commitTransaction();
```

In doing so, the report's state change to "finalized" and it's saved in the locally stored data. Updating the remote database may therefore be done downstream of these operations based on the local changes.

Realm's features list does not stop there. This library provides, among other, primary keys management - used to manage the relationships between objects, allows a lot of queries, supports JSON, facilitates the migration of the local database and even comes with a program used to consult the database through a practical and ergonomic interface.

[157] cf. 5.7.7 Intent

7 Comments and suggestions

Ethias Prevention is ultimately a modest application. Very useful to Ethias field agents who are the target audience, it will go into production soon, that is to say, it will be accessible to users, and will join its iOS equivalent among the commercial apps. It is ergonomic and generally well thought, not to mention the explicit and efficient design in regard to its operating, useful for not having to train field staff about its use. However, its implementation was not easy, because, among others, of the possibility to add an infinite number of sections to a report and an infinite number of observations within each section which was to be implemented accordingly to the MVVM. It is within this overall perspective that are here thought possible improvements in regard of the app's development, either from the standpoint of the features or of the running of the development.

The first and easiest comment to make about this project is that there is still no Windows Phone version of the app. Wanting to target the widest public is equivalent to having to develop on as many platforms as possible which means, in the mobile world that it's necessary to not forget users equipped with Windows smartphones, third on the podium of the major mobile operating systems.

A word on the development of the Android version seems necessary as well. Google's Data Binding library[158] is full of good intentions and undeniably practical features, but it is still very young and there still remains a rather significant number of cases where its use will help little in regard of the MVVM design pattern.[159] Thus it is appropriate in these times to improvise at best by writing code that often only workaround or is of a laughable size compared to what it accomplishes. This generates consequent losses of time. Perhaps the MVC and the MVP[160] still have some good years ahead of them, until Google's library gets better thanks to the interventions of the community.

In terms of functionalities, it should be said that Ethias Prevention is pretty complete in regard of its scope. It is difficult to do better as it supports everything from the creation of a report to its finalization with the generation, the sending by mail and the printing of a PDF version of those reports and it can even enrich them with media and assist the user with proposals as to the nature of the tasks and observations. It also allows geolocation in order to avoid the pain of having to manually enter the place of intervention and it also uses the day of creation of the report as the visiting date by default. It is even possible to introduce pictograms illustrating the risks if necessary.

[158] cf. 5.7.11.2 The Data Binding library
[159] cf. 5.7.11.1 Presentation of the MVVM
[160] cf. 5.7.11.1 Presentation of the MVVM

Despite this, it is easy to imagine even more features once the rather limited scope of Ethias Prevention is set aside. The identification at the app launch suggests, for example, the consideration of a section centered on the management of an embedded desktop. Such a dedicated area would allow the agent to read emails, for example, or his schedule for the day or even the week including the different interventions and prevention tasks he must perform. The whole administrative management of his account could be performed here for a reduced presence in the offices of Ethias and therefore an increased presence on the field with obvious advantages. Involving a more consistent development but susceptible to generate some advantages, can also be considered adding a gateway to the Ethias intranet and to all its features.

Regarding the internship, its course happened without pitfall. The first two weeks dedicated to creating a fictitious app were a perfect training about the ins and outs of developing on Android and it's hard to imagine a better introduction. It was enough afterwards to just have to do some additional research in parallel of the production to achieve a complete and professional app that is up to the standard of the current applications. The few tools[161] used at Djm digital were very useful to the comfort of development, especially SourceTree, a tool for centralizing and reviewing code. The quality of the materials provided, however, was a notable obstacle given the fact that the writing and the testing of the code mainly occurred on an older computer with an almost obsolete version of Windows. This had consequences for long builds during which the only possibility was to wait for it to finish, causing an increase in the total time required to deploy the app. Also, the limited number of available machines used to test the application could have been increased to ensure the absence of any unexpected behavior.

Communication, meanwhile, is also a candidate for improvement. Indeed, the iOS version of Ethias Prevention already being done, it served as indispensable support to the development of its Android equivalent but the provision of analysis upstream to clarify certain details would have been welcome. Also, due to the lack of experience, a number of questions had to be asked. This was often made spontaneously generating perhaps sometimes a waste of time for the speaker or the listener. Organizing meetings or using a dedicated communication software could have optimized these interactions.

Finally, to conclude these criticisms and suggestions, it is good to focus briefly on the platform that runs Ethias Prevention, namely Android.[162] Although boasting a faithful and dedicated community of developers and users, Google's mobile operating

[161] cf. 4 Processes, tools
[162] cf. 5 Android

system is open source, which tends to cause internal consistency problems. Also, Google is known not to be a very well organized company in regard of its products. Thus, incorporating features such as media management, geolocation, printing PDF and others is therefore more complex than on a generally more stable platform like iOS or Windows Phone. The issue of backward compatibility is also problematic despite the support brought by Google[163] and each announcement of a new feature is usually preceded by a sweat from the developers. However, Android is still a powerful multitasking platform that should be considered for each app development.

[163] cf. 5.7.10 Android support library

8 Conclusion

Twelve weeks have been devoted to achieving Ethias Prevention and there are still some tasks to accomplish before releasing it. Managing different languages, implementing a solution on Ethias side in regard of the forgotten password or doing a little design makeover are many tasks that are still to be done before the export of the application on Google Play. But actually, Ethias Prevention is now usable and the development of its main features is completed. It is therefore fashionable, as a final word for this report to lean back on the experience lived during the last few months.

It is in a professional but friendly atmosphere that this internship took place. The first two weeks were dedicated to my training in Android, rather effective, through a direct immersion in the development of a mobile app, and allowed me to familiarize with the business world. The twelves following weeks were entirely devoted to Ethias Prevention but were nonetheless rewarding and exciting. The work was done well and I learned a lot, I could even almost say that the development on Android has no secrets for me anymore.

This immersion in the business world also taught me a lot about the organization of the work that is done there. For example, I could attend the conduct of an ambitious project concerning connected kickers as well from a development standpoint as from a marketing one, opening my eyes to the complexity of the working of such an operation. I also got a glimpse of the rigor asked in the field, which allowed me to go over my own work methods in order to improve them. If some points are still candidates for improvement such as corporate communication and the provided material, the experience was still rich. Being responsible for the development of an app that will be useful to many people is something magical of which I can only be proud.

There's one thing that is sure after this immersion phase in the business world, it is that information technologies are still in their teenage and that they have a beautiful future. I do not regret my choice to specialize in this area and continue to consider the path of software development in regard of my career.

9 BIBLIOGRAPHY

Acesyde. (2012, Novembre 23). *Accéder aux services web via Android*. Retrieved Mai 18, 2016, from Developpez.com: http://acesyde.developpez.com/tutoriels/android/web-services-android/

Activity | Android Developers. (2016, Mai 04). Retrieved Mai 09, 2016, from Android Developers: http://developer.android.com/reference/android/app/Activity.html

ADAM, L. (2015). *Analyse et développement d'applications mobiles dédiées aux appareils sous iOS.* Travail de fin d'étude, Haute Ecole de Namur-Liège-Luxembourg - Implantation IESN, Namur. Retrieved Mai 18, 2016

Adapter | Android Developers. (2016, Mai 04). Retrieved Mai 09, 2016, from Android Developers: http://developer.android.com/reference/android/widget/Adapter.html

AMADEO, R. (2012, Septembre 17). *A History of Pre-Cupcake Android Codenames*. Retrieved Février 23, 2016, from Android Police: http://www.androidpolice.com/2012/09/17/a-history-of-pre-cupcake-android-codenames/

Android. (n.d.). Retrieved Février 23, 2016, from Android: http://www.android.com/

Android Developers. (n.d.). Retrieved Mai 17, 2016, from Android Developers: https://developer.android.com/index.html

Android Interfaces and Architecture | Android Open Source Project. (2016, Février 25). Retrieved Février 23, 2016, from Android Open Source Project: http://source.android.com/devices/

Android Pugin for Gradle | Android Developers. (2016, Février 19). Retrieved Février 23, 2016, from Android Developers: http://developer.android.com/tools/building/plugin-for-gradle.html

Android versions comparison | Comparison tables - SocialCompare. (2016, Février 16). Retrieved Février 23, 2016, from SocialCompare: http://socialcompare.com/en/comparison/android-versions-comparison

App Manifest | Android Developers. (n.d.). Retrieved Mai 12, 2016, from Android Developers: http://developer.android.com/guide/topics/manifest/manifest-intro.html

Application | Android Developers. (2016, Mai 04). Retrieved Mai 09, 2016, from Android Developers: http://developer.android.com/reference/android/app/Application.html

BANERJEE, A. (2014, Juin 25). *Android Auto unveiled at Google I/O: what you need to know | AndroidAuthority*. Retrieved Février 23, 2016, from AndroidAuthority: http://www.androidauthority.com/android-auto-google-io-397118/

Bibliothèque d'API. (n.d.). Retrieved Mai 22, 2016, from Console Developers: https://console.developers.google.com/?pli=1

BIRCH, J. (2015, Septembre 21). *Approaching Android with MVVM -- ribot labs*. Retrieved Mai 10, 2016, from ribot labs: https://labs.ribot.co.uk/approaching-android-with-mvvm-8ceec02d5442#.2vbsqode9

BYOUS, J. (2003, Avril). *Java Technology: The Early Years*. Retrieved Février 25, 2016, from Sun microsystems:

http://web.archive.org/web/20100105045840/http://java.sun.com/features/1998/05/birthday.html

CAMPBELL, A. (2015, Novembre 15). *Android Data Binding Tutorial*. Retrieved Mai 17, 2016, from CapTech: https://www.captechconsulting.com/blogs/android-data-binding-tutorial

Context | Android Developers. (2016, Mai 04). Retrieved 04 09, 2016, from Android Developers: http://developer.android.com/reference/android/content/Context.html

DANON, G. (2014, Juin 13). *Les machines à laver Samsung et leurs application*. Retrieved Février 23, 2016, from Android-France: http://android-france.fr/2014/06/les-machines-laver-samsung-application/

Dashboards | Android Developers. (2016). Retrieved Février 23, 2016, from Android Developers: http://developer.android.com/about/dashboards/index.html

Data Binding Guide | Android Developers. (n.d.). Retrieved Mai 10, 2016, from Android Developers: http://developer.android.com/tools/data-binding/guide.html#build_environment

Djm Digital : Création de site internet & application mobile. (n.d.). Retrieved Février 18, 2016, from djmdigital: http://www.djmdigital.be/

D'ORAZIO, D. (2014, Mars 18). *Google reveals Android Wear, an operating system for smartwatches | TheVerge*. Retrieved Février 23, 2016, from TheVerge: http://www.theverge.com/2014/3/18/5522226/google-reveals-android-wear-an-operating-system-designed-for

Download Android Studio and SDK Tools | Android Developers. (n.d.). Retrieved Février 18, 2016, from Android Developers: http://developer.android.com/sdk/index.html

DUBISY, F. (2015). *Programmation mobile.* Namur: Haute École de Namur - Liège - Luxembourg Implantation IESN.

Gestures - Patterns - Google design guidelines. (n.d.). Retrieved Mai 09, 2016, from Google: https://www.google.com/design/spec/patterns/gestures.html#gestures-touch-mechanics

Getting Started with Data Binding in Android. (2015, Septembre 15). Retrieved Mai 10, 2016, from opgenorth.net: http://www.opgenorth.net/blog/2015/09/15/android-data-binding-intro/

Intent | Android Developers. (2016, Mai 04). Retrieved Mai 09, 2016, from Android Developers: http://developer.android.com/reference/android/content/Intent.html

Introduction - Material Design - Google design guidelines. (n.d.). Retrieved Février 25, 2016, from Material Design: https://www.google.com/design/spec/material-design/introduction.html

Java Docs - Realm is a mobile database: a replacement for SQLite & Core Data. (2016). Retrieved Mai 18, 2016, from Realm: https://realm.io/docs/java/latest/

JSON. (n.d.). Retrieved Mai 18, 2016, from json.org: http://www.json.org/

LARDINOIS, F. (2015, Mai 28). *Google Launches Android M Preview With Fingerprint Scanner Support, Android Pay, Improved Permissions And Battery Life | TechCrunch*. Retrieved Février 23, 2016, from TechCrunch: http://techcrunch.com/2015/05/28/google-announces-android-m-with-fingerprint-scanner-support-android-pay-improved-permissions-battery-and-performance-tweaks/

LEGGETT, J. (2011, Avril 8). *Android timeline 2003-2011*. Retrieved Février 23, 2016, from USwitch: http://www.uswitch.com/mobiles/news/2011/04/android-timeline-2003-2011/

Main Page - Android Wiki. (2013, Août 5). Retrieved Février 18, 2016, from Android Wiki: http://android-dls.com/wiki/index.php?title=Main_Page

MAZELIER, G. (2011, Septembre). *Build automatisé : à la découverte de Gradle / GLMF-141 / Linux Magazine / GNU / Connect - Edition Diamond*. Retrieved Février 19, 2016, from Connect - Edition Diamond: http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-141/Build-automatise-a-la-decouverte-de-Gradle

MERCHIE, V. (2013). *Mise à jour d'une application mobile Android : "Belfius Travel".* Travail de fin d'étude, Haute École de la Province de Liège, Liège. Retrieved Février 27, 2016

OPAM, K. (2014, Juin 25). *Google officially unveils Android TV | TheVerge*. Retrieved Février 23, 2016, from TheVerge: http://www.theverge.com/2014/6/25/5840424/google-announces-android-tv

Oracle and Sun Microsystems | Strategic Acquisitions | Oracle. (n.d.). Retrieved Février 25, 2016, from Oracle: https://www.oracle.com/sun/index.html

Premiers pas | Google Maps Android API | Google Developers. (2016, Mars 24). Retrieved Mai 22, 2016, from Google Developers: https://developers.google.com/maps/documentation/android-api/start#etape_4_obtenir_une_cle_dapi_google_maps

Retrofit. (n.d.). Retrieved Mai 18, 2016, from square.github.io: http://square.github.io/retrofit/

Services | Android Developers. (n.d.). Retrieved Mai 09, 2016, from Android Developers: http://developer.android.com/guide/components/services.html

Signature et clés d'API | Google Maps Android API | Google Developpers. (2016, Mars 24). Retrieved Mai 22, 2016, from Google Developers: https://developers.google.com/maps/documentation/android-api/signup#release-cert

Suport Library Features | Android Developers. (2016, Mai 09). Retrieved from Android Developers: http://developer.android.com/tools/support-library/features.html#v4

Supporting Multiple Screens | Android Developers. (2016, Mai 09). Retrieved from Android Developers: http://developer.android.com/guide/practices/screens_support.html

Torrent file - Wikiwand. (n.d.). Retrieved Mars 13, 2016, from Wikiwand: https://www.wikiwand.com/en/Torrent_file

Unit Testing. (2016). Retrieved Mai 10, 2016, from Microsoft | Developer: https://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx?f=255&MSPPError=-2147217396

ViewGroup | Android Developers. (n.d.). Retrieved Mai 17, 2016, from Android Developers: https://developer.android.com/reference/android/view/ViewGroup.html

WALSH, N. (1998, Octobre 03). *A Technical Introduction to XML*. Retrieved Mai 10, 2016, from XML.com: http://www.xml.com/pub/a/98/10/guide0.html

ZDNet.fr, L. r. (2015, Décembre 7). *Chiffres clés : les OS pour smartphones*. Retrieved Février 23, 2016, from ZDNet.fr: http://www.zdnet.fr/actualites/chiffres-cles-les-os-pour-smartphones-39790245.htm

10 TABLE OF FIGURES

11 ANNEXES

11.1 A PDF REPORT

ethias PREVENTION

RAPPORT DE VISITE

Dossier	Folder name
Référence	123/456
Client	Client Name
Date de visite	18-03-16
Agent	Michele Obama
Type	Dépistage de risques - Vol ou dégradations
Lieu	Rue Haute, 4600 Visé
Participants	Participants
Destinataires	Distribution

Préliminaire

Le présent rapport est un compte-rendu des observations concrètes faites sur les lieux de travail et formule des recommandations dans le cadre d'une démarche d'amélioration de la sécurité et du bien -être des travailleurs dans l'entreprise.

Ce rapport constitue un instantané et ne doit donc pas être considéré comme exhaustif. La mise en œuvre des démarches d'améliorations relève de la responsabilité de l'employeur et des membres de la ligne hiérarchique (article 5 de la loi du 4/08/1996 relative au bien-être au travail et article 13 de l'AR du 27 mars 1998 portant sur les principes généraux relatifs à la politique du bien-être au travail).

Observations générales

Observations

Recommandations générales

Recomendations

ethias PREVENTION

Section 1

Section 1 – Topic 1

Risques				Priorité			
Ok	Faible	Modéré	Haut	Basse	Moyenne	Elevée	Urgente

Catégories

Législation(s)

Pictogrammes

Commentaires

Section 1 – Topic 1 – Observations