

# **A mobile application for weather and pollen statistics**

industriële wetenschappen en technologie  
bachelor in de elektronica-ICT

Eindwerk aangeboden tot het  
behalen van het diploma van  
bachelor in de elektronica-ICT

door Milan Lamote

Campus Oostende

Academiejaar 2016 - 2017

o.l.v. Tom Cordemans, Vives

Luc Vanhee, Vives

Quiliano Isaac Moro Sancho, E. Ingeniería Informática de Valladolid

Javier Bastida Ibáñez, E. Ingeniería Informática de Valladolid



# **A mobile application for weather and pollen statistics**

industriële wetenschappen en technologie  
bachelor in de elektronica-ICT

Eindwerk aangeboden tot het  
behalen van het diploma van  
bachelor in de elektronica-ICT

door Milan Lamote

Campus Oostende

Academiejaar 2016 - 2017

o.l.v. Tom Cordemans, Vives

Luc Vanhee, Vives

Quiliano Isaac Moro Sancho, E. Ingeniería Informática de Valladolid

Javier Bastida Ibáñez, E. Ingeniería Informática de Valladolid

## Announcement

This final project was an exam. The questions formulated during the defense presentation are not included.

Deze eindverhandeling was een examen. De tijdens de verdediging geformuleerde opmerkingen werden niet opgenomen.

## Abstract in English

This bachelor's thesis was given to me by the 'Universidad de Valladolid', it is about an Android application for collecting and showing weather statistics. These statistics concern things like temperature, wind, rainfall and pollen information from the current day as well as the next few days. The goal is to be able to download the needed information, save it and show it to the user in a way he or she can read it. This book will cover the whole process from scratch to how to install. The interface will be explained using a graphical user manual.

## Abstract in Dutch

Dit eindwerk werd mij aangeboden door de 'Universidad de Valladolid', het betreft een Android applicatie voor het verzamelen en tentoonstellen van weerstatistieken. Deze statistieken betreffen informatie zoals de temperatuur, de wind, de neerslag en stuifmeelconcentratie van de huidige dag zowel als enkele komende dagen. Het doel is om deze informatie te downloaden, op te slaan en in een leesbaar formaat aan de gebruiker te tonen. Dit boek omvat het hele proces van nul tot hoe te installeren. Het gebruikersscherm wordt toegelicht aan de hand van een gebruikershandleiding.

## Abstract in Spanish

Esta tesis de licenciatura ha sido realizada en la Universidad de Valladolid, se trata de una aplicación de Android para recopilar y mostrar estadísticas del tiempo. Estas estadísticas se refieren a cosas como la temperatura, el viento, la lluvia y la información sobre el polen del día actual, así como los próximos días. El objetivo es poder descargar la información necesaria, guardarla y mostrarla al usuario de manera que pueda leerla. Este documento cubrirá todo el proceso desde cero, hasta cómo instalarlo. La interfaz se explicará mediante un manual de usuario gráfico.

## Dedication in English

I would like to thank 'VIVES University College' for giving me the opportunity of completing my studies abroad with an Erasmus+ program. By this, I'd like to personally thank my internal promotors Cordemans Tom and Vanhee Luc. They've prepared me in any way possible for this experience. Although quite the distance they were always prepared to answer any questions I had via mail. Secondly, I'd like to thank the 'Universidad de Valladolid' for accepting and helping me through this journey. By this, I'd like to thank my external promotor Quilano Isaac Moro Sancho and supervising professor Javier Bastida Ibáñez. I was always welcome at their offices for any questions or further ideas concerning my project. Thirdly, I'd like to thank my local student mentor and friend Tamara Alonso Bernal. She has helped me a lot getting settled in and giving me opportunities to get to know the city and meeting new friends. Lastly, I'd like to thank my family for supporting me in taking on this project abroad. I wouldn't have been able to enjoy this experience as much as I did without their support. This project has taught me a lot on how to take on a big project by myself even in an unknown environment, an environment most likely different than the one I will be spending my career in. During the process, I've made a lot of friends and memories, things I will never forget and will be able to use in my further life.

## Dedication in Dutch

Ik wil graag 'Hogeschool VIVES' bedanken om mij de gelegenheid te geven mijn studies in het buitenland te voltooien met een Erasmus+ programma. Hierbij wil ik persoonlijk mijn interne promotors Cordemans Tom en Vanhee Luc bedanken. Zij waren een grote hulp bij mijn voorbereiding op dit avontuur. Alhoewel ze niet nabij waren, waren ze steeds bereid om vragen via mail te beantwoorden. Ten tweede wil ik de 'Universidad de Valladolid' bedanken om mij als student te accepteren en mij te helpen doorheen dit avontuur. Hierbij bedank ik mijn externe promotor Quilano Isaac Moro Sancho en begeleidende professor Javier Bastida Ibáñez. Ik was altijd welkom bij hun op kantoor voor vragen of verdere ideeën in verband met mijn project. Ten derde wil ik mijn lokale student-mentor en vriend Tamara Alonso Bernal bedanken. Zij heeft mij vooral geholpen me te vestigen daarnaast gaf ze mij kansen de stad te leren kennen en nieuwe vrienden te ontmoeten. Tenslotte wil ik mijn familie bedanken voor de steun bij mijn keuze om dit project in het buitenland uit te voeren. Ik had nooit zoveel van deze ervaring kunnen genieten zonder hun hulp. Dit project heeft mij vooral veel geleerd over hoe ik een groot project dien aan te pakken. Zelf in een onbekende omgeving, een omgeving die waarschijnlijk anders is dan die waar ik de rest van mijn carrière zal doorbrengen. Tijdens dit proces heb ik veel vrienden en herinneringen gemaakt dingen die ik nooit zal vergeten en ik in mijn verdere leven zal kunnen gebruiken.

## Table of contents

|  |    |
|--|----|
| Announcement.....                      | 4  |
| Abstract in English.....               | 4  |
| Abstract in Dutch .....                | 4  |
| Abstract in Spanish.....               | 4  |
| Dedication in English .....            | 5  |
| Dedication in Dutch.....               | 5  |
| Table of contents .....                | 4  |
| List of figures.....                   | 6  |
| Objectives.....                        | 7  |
| Introduction .....                     | 8  |
| Planning.....                          | 10 |
| February .....                         | 10 |
| March.....                             | 10 |
| April.....                             | 11 |
| May.....                               | 11 |
| 1 Project's logic .....                | 12 |
| 1.1 Gather sources.....                | 12 |
| 1.1.1 Past and current data.....       | 13 |
| 1.1.2 Predictions .....                | 14 |
| 1.1.3 Pollen .....                     | 14 |
| 1.2 Download data .....                | 16 |
| 1.2.1 Connection.....                  | 16 |
| 1.2.2 DownloadManager .....            | 16 |
| 1.2.3 BroadcastReceiver .....          | 17 |
| 1.3 Read data.....                     | 19 |
| 1.3.1 CSVReader.....                   | 19 |
| 1.3.2 xmlParser .....                  | 20 |
| 1.4 Show data.....                     | 23 |
| 1.4.1 Swipe pages.....                 | 23 |
| 1.4.2 TextView, ImageView.....         | 23 |
| 1.4.3 PopupWindow, Radio buttons ..... | 23 |
| 1.4.4 ScrollView .....                 | 24 |

|       |                             |    |
|-------|-----------------------------|----|
| 1.4.5 | GridView .....              | 24 |
| 2     | Logic's implementation..... | 25 |
|       | Manifest.....               | 25 |
| 2.1   | Sources.....                | 26 |
|       | Selecting location.....     | 27 |
|       | URLs.....                   | 29 |
| 2.2   | Downloading .....           | 30 |
| 2.2.1 | Connection.....             | 32 |
| 2.2.2 | DownloadManager .....       | 32 |
| 2.2.3 | BroadcastReceiver .....     | 34 |
| 2.3   | Read data.....              | 35 |
|       | Weather class.....          | 35 |
|       | Pollen class.....           | 37 |
| 2.3.1 | CSV reader .....            | 38 |
| 2.3.2 | Xmlparser.....              | 40 |
| 2.4   | Show data.....              | 43 |
| 2.4.1 | Swipe pages.....            | 45 |
| 2.4.2 | ScrollView .....            | 51 |
| 2.4.3 | GridView .....              | 52 |
| 3     | Tests .....                 | 53 |
| 3.1   | Finished downloads.....     | 53 |
| 3.2   | Reading the files .....     | 53 |
| 3.3   | Background image.....       | 54 |
| 3.4   | X-axis labels.....          | 54 |
| 4     | Conclusion .....            | 55 |
| 5     | Appendix .....              | 56 |
| 5.1   | Manual.....                 | 56 |
|       | Installation .....          | 56 |
|       | How to use .....            | 58 |
| 5.2   | Bibliography .....          | 62 |

## List of figures

|   |    |
|---|----|
| Figure 1: Smartphone OS market share .....        | 8  |
| Figure 2: Historic data, website .....            | 13 |
| Figure 3: Predictions, website .....              | 14 |
| Figure 4: XmlPullParser explanation .....         | 20 |
| Figure 5: GraphView XML layout .....              | 24 |
| Figure 6: GraphView code snippet.....             | 24 |
| Figure 7: GraphView example .....                 | 24 |
| Figure 8: Sources flowchart.....                  | 26 |
| Figure 9: Edit location screenshots .....         | 27 |
| Figure 10: Edit location snippet .....            | 28 |
| Figure 11: Constant UML.....                      | 29 |
| Figure 12: MainActivity UML.....                  | 30 |
| Figure 13: Downloading flowchart .....            | 31 |
| Figure 14: Network state manifest.....            | 32 |
| Figure 15: Access Shared Preferences snippet..... | 32 |
| Figure 16: File location snippet.....             | 33 |
| Figure 17: Weather UML.....                       | 35 |
| Figure 18: Pollen UML.....                        | 37 |
| Figure 19: CSV file example.....                  | 38 |
| Figure 20: csvParser UML.....                     | 39 |
| Figure 21: XML file example.....                  | 40 |
| Figure 22: xmlParser UML.....                     | 41 |
| Figure 23: XML tags snippet .....                 | 41 |
| Figure 24: KEY_PROBPRECI snippet .....            | 42 |
| Figure 25: MainActivity UML.....                  | 43 |
| Figure 26: Data showing flowchart.....            | 44 |
| Figure 27: Fragment instances snippet .....       | 45 |
| Figure 28: Fragment class 1, layout 1 .....       | 46 |
| Figure 29: Fragment class 2, layout 1 .....       | 47 |
| Figure 30: Fragment class 3, layout 2 .....       | 47 |
| Figure 31: FirstFragment UML.....                 | 48 |
| Figure 32: Set fragment arguments snippet .....   | 48 |
| Figure 33: Get fragment arguments snippet.....    | 49 |
| Figure 34: Resize background image .....          | 49 |
| Figure 35: ScrollView screenshots.....            | 51 |
| Figure 36: graphActivity UML .....                | 52 |
| Figure 37: Device usage screenshot.....           | 56 |
| Figure 38: Installation screenshots .....         | 57 |
| Figure 39: Open application screenshot.....       | 58 |
| Figure 40: Swipe screenshots.....                 | 59 |
| Figure 41: Select location screenshots .....      | 60 |
| Figure 42: Pollen screenshot.....                 | 60 |
| Figure 43: Charts screenshots .....               | 61 |



## Objectives

Before we begin, it's important to know what this project is about and what its objectives are. This project is in fact a mobile application which gathers and displays information about weather and pollen statistics. In order to do so we have some objectives we need to fulfill. The following list summarizes these objectives.

### Objectives

**Gathering the right sources where to obtain the necessary data.  
This data contains information concerning:**

- **Past 24 hour weather statistics**
- **Current weather status**
- **Weather predictions**
- **Pollen status**

**Downloading this data to the user's device in order to be able to let the application handle this data.**

**Read the data collected from the internet and contain it within the application.**

**Present this data to the user in an easy to read manner.**

---

# Introduction

When we try to find information about weather statistics there are a lot of different places to look. We can look it up online, in the newspaper, on the television and so on. In this work, I'm focusing on the information found online. This information is most of the time scattered amongst different pages and it takes some time to find everything you need. And, even though everything can be found online, it isn't always easy to read or even easy to access.

This application is in some way a collector of all this information and makes it easier for the user to find the information needed in just one place, the application. It provides a clear view of all the information it has gathered and shows it so the user can access it all in one touch. It's easily read and provides a lot of information with just one glance at the screen.

This work concerns the process of making this application. It explains the things you need as well as how to use these tools in order to produce such an application.

This project was given to me by the 'Universidad de Valladolid', they've given me just the idea of an application which contains information about the weather and makes it easy accessible for everyone. Based on this idea I started brainstorming on how to make this happen.

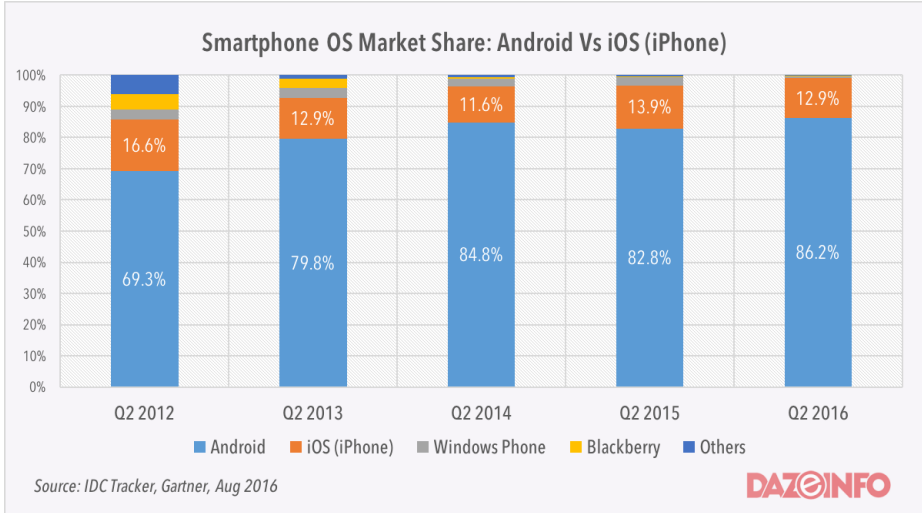


Figure 1: Smartphone OS market share

The first question I've encountered was the choice of environment I'd be working with. Based not only on my own preference but based on worldwide statistics we can see that Android keeps growing in the market share. At this point around 86% of all mobile devices are running with the Android operating system based on the data in figure 1. (1)

At second place, we have the Apple operating system. Although also used in up to 13% of all the devices in the world, it didn't struck my interest for the following reasons. I didn't have any experience with the Apple development environment. I did look it up but noticed that if you want a complete environment it will cost you € 99/year (2). Even if you use free environments they are at first very limited and Apple provides a lot of rules and guidelines you are obligated to follow.

Based on these facts I chose for the Android environment. It's free to use and everything is included into the development environment, Android Studio. Some other fun facts are that Android is opensource which makes it easy to do research on how to develop, it provides a lot of libraries to tackle whichever problem and it contains an online user manual where every class, method or function is described in an understandable way.

So, the winner was Android. "Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. [...] Android's source code is released by Google under an open source license. [...] Its open nature has encouraged a large community of developers and enthusiasts (such as myself) to use the open-source code as a foundation for community-driven projects." (3)

With the environment chosen, I could start working on the project itself. The first thing to do was to find the different places online where I would fetch my information. My mentor Isaac suggested me with the page 'aemet.es', it's a website where I could find a lot of information and use this information without any sort of login required. This was what I needed because I found other options where I could use an API, but for this, I was in need of an API key. Which was free to get but when someone else wanted to use my application they would have to make an account themselves and pass their own API key in order for the application to keep working as it should. Thanks to 'aemet.es' this was not the case. The information found on this website could be obtained through CSV- and XML-files.

I've made the decision to save these files locally on the users device. This way when the user does not have access to the internet they can still see the information gained from the last time they were connected to the internet. The files itself do not take a lot of space on the device and are stored as files from the application so the user does not have to, but can if they want to, interfere with them.

Once the files are on the device, they are immediately processed. Since there are two types of files, CSV and XML, this project contains two different parsers for each of these types.

Due to this data originating from a Spanish website, the information gained is only for cities in Spain. Nonetheless, it is easy to expand the recourses of this project for other countries if you have valuable sources for this information.

## Planning

To make this final project and write the thesis I had a timespan of four months. In February I started working on the project and by the 31<sup>st</sup> of May I had to present my work. During these four months I had to prepare, do research, develop and test the application, write this thesis and prepare my defense.

### February

During this month I spent most of my time exploring the possible sources. I've got some suggestions by my mentor and I found some options myself too. In the end these options I found did not seem to be a good solution because they were not entirely free to use. Therefore I chose to move on with the sources I've gotten from my mentor.

I then started to refresh my knowledge of the Android environment. I followed some online courses to make a basic application. I reread some of my books from the past years in college.

Once I was accustomed with the surroundings, I started building the first version of my application. The first thing I had to do was getting the online information on my device. I experimented with different methods in order to directly read the file and process it. But later on decided it's better to store the data on the device in case there is no connection the user still has the chance to see some saved information.

Once I was able to download these files I tried to do a simple parsing action on them but that didn't work at first, there was no data found while I could see the file located on the device.

### March

I've solved the parsing problem by implementing a broadcast receiver which told me when the download was finished. After which I tried parsing again, this time successfully.

I implemented different ways of representing the data I've collected. I made some simple lists, tried to fill some tables and even make some charts with the data I've obtained.

Then I've improved the downloader to download more than one file, which was now the case. I was able to download information about the current state, predictions and pollen.

The application at this point was able to download and display the data but it wasn't very beautiful or easy in use. So I started thinking of ways to improve the design.

## April

It was a long struggle to find a good and suitable layout I'd be satisfied with myself. But I did find it, I got the idea to make separate pages for each day and make the user swipe through them.

In order to do so it was easier to start with a fresh application, nonetheless I could still use the knowledge from my first application. It took me some time to get everything in a good place I liked, but I managed to finish the design on my device. But, when I tried my application on a different device it didn't look very good. Therefore I had to redo some of the application in order to make it more flexible.

I also implemented different locations into the application, this would make the application applicable for a wider range of people.

## May

Once I had my pages swiftly working I started building 2 last activities, one for historic data and one for the pollen. For these I mostly used parts from my first application.

I did have a hard time reading the XML file containing information about the pollen. The parser returned faulty information although this parser was mostly recycled from the one used to read the predictions, which worked perfectly. After a long search and building the parser part by part I found out I forgot a statement and wrote some typo's. Although very frustrating I did feel very relieved once I had sorted this out.

This month I also built up my thesis, which was more work than expected. But I did have a good list of sources which I used during the process, this list did help me a lot by refreshing the different steps I went through.

# 1 Project's logic

In order to take this project to a good finish we need to achieve some goals:

- Gather sources
- Download data
- Read data
- Present data to user

This chapter will be covering how we managed to complete these objectives and what logic we've used to do so. It gives an explanation on what methods and what knowledge we've used in order to achieve each of these goals. While in the next chapter we will apply these methods and knowledge in order to make the application itself.

## 1.1 Gather sources

First things first, we need to find good and reliable sources to obtain our information. Information about the current weather state, the weather predictions and pollen. With the fact that this work was made while in Valladolid, Spain, the information we use concerns the Spanish weather.

For the first three (past statistics current state and predictions) we make use of the website [www.aemet.es](http://www.aemet.es). It's a Spanish website which stands for 'Agencia Estatal de Meteorología' or 'State Agency of Meteorology'. It provides us with information about the weather, current status as well as predictions.

The last one, the information about the pollen, can be found on the website [www.datosabiertos.jcyl.es](http://www.datosabiertos.jcyl.es) which stands for open data which originates from 'Junta de Castilla y León'.

"The Junta of Castile and León (Spanish: *Junta de Castilla y León*) is the governing and administrative body of the Spanish autonomous community of Castile and León and serves as the executive branch and regulatory authority. [...] The function of the Junta is to govern and administer the autonomous community." (4)

We can now dive deeper into each of these sources and show you where we found each part of the information necessary to build up this project.

### 1.1.1 Past and current data

Concerning the past statistics and current status we can find data which is updated every hour for different cities in Spain. We find information about temperature, wind, precipitation, air pressure and humidity. For example, the data for Valladolid can be found at:

<http://www.aemet.es/es/eltiempo/observacion/ultimosdatos?k=cle&l=2422&w=0&datos=det&x=h24&f=temperatura>

| Fecha y hora oficial | Temp. (°C) | V. vien. (km/h) | Dir. viento | Racha (km/h) | Dir. racha | Prec. (mm) | Presión (hPa) | Tend. (hPa) | Humedad (%) |
|----------------------|------------|-----------------|-------------|--------------|------------|------------|---------------|-------------|-------------|
| 17/05/2017 16:00     | 15.5       | 9               | →           | 32           | ↓          | 1.4        | 934.7         | 1.1         | 91          |
| 17/05/2017 15:00     | 16.7       | 13              | ↘           | 27           | ↘          | 0.2        | 934.4         | 0.9         | 85          |
| 17/05/2017 14:00     | 19.5       | 13              | ↘           | 27           | ↘          | 0.0        | 933.9         | 0.3         | 64          |
| 17/05/2017 13:00     | 21.8       | 9               | →           | 21           | →          | 0.0        | 933.6         | -0.1        | 60          |
| 17/05/2017 12:00     | 22.3       | 8               | ↗           | 23           | →          | 0.0        | 933.5         | -0.1        | 46          |
| 17/05/2017 11:00     | 20.5       | 7               | ↑           | 21           | ↑          | 0.0        | 933.6         | -0.4        | 43          |
| 17/05/2017 10:00     | 20.0       | 5               | ↑           | 24           | ↑          | 0.0        | 933.7         | -0.3        | 47          |
| 17/05/2017 09:00     | 20.0       | 9               | ↑           | 19           | ↑          | 0.0        | 933.6         | -0.4        | 46          |
| 17/05/2017 08:00     | 17.8       | 4               | ↑           | 15           | ↑          | 0.0        | 934.0         | -0.3        | 59          |
| 17/05/2017 07:00     | 17.8       | 6               | ↑           | 19           | ↑          | 0.0        | 934.0         | -0.5        | 63          |

Figure 2: Historic data, website

This information is also available for download in two different formats, XLS and CSV. The XLS-format is specifically for Microsoft Excel while the CSV-format provides us with a plain text file containing the information in such a way that its more or less readable by a human as well. Since our application is designed for Android, we prefer to use the CSV-format because this makes it possible to read and process the contents of the file directly.

### 1.1.2 Predictions

Next, we have information about weather predictions which we can also find on this website. We find information about temperature, precipitation, wind and the state of the sky. This information is also available for multiple Spanish cities. For example, the data for Valladolid can be found at:

<http://www.aemet.es/es/eltiempo/prediccion/municipios/valladolid-id47186>

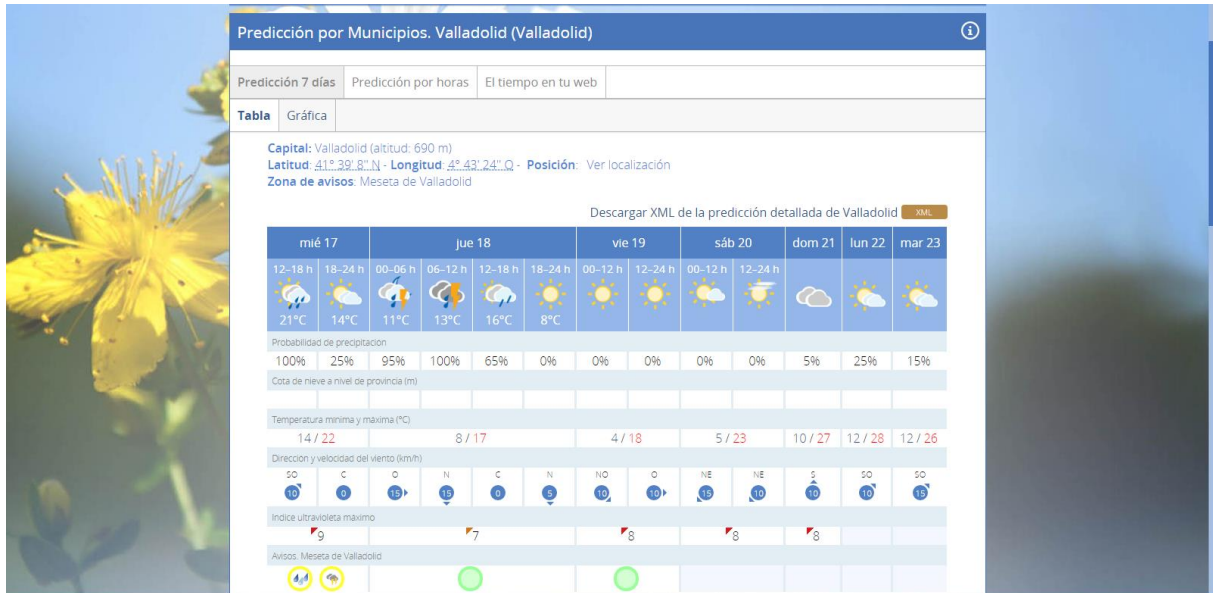


Figure 3: Predictions, website

This information can be downloaded as a file with an XML-format. This format is also a plain text file readable by a human being with possibilities to read and process in our application.

### 1.1.3 Pollen


Last but not least, we need some information about pollen. Pollen consist of male cells that fertilize plants, these are richly present during some periods. Pollen are often a big cause for allergic reactions. We want to provide the user with information concerning the concentration of pollen present.

The data provided is available for download in two different formats, CSV or XML. We choose the XML-format because this makes it easier for us to read the data; this will be clarified in another chapter. This file provides us with data from 13 different cities and 36 different types of pollen. The specific link is

[http://www.datosabiertos.jcyl.es/web/jcyl/set/es/mediciones/niveles\\_de\\_polen/1284208096554](http://www.datosabiertos.jcyl.es/web/jcyl/set/es/mediciones/niveles_de_polen/1284208096554)



We do now have all the sources we need in order to retrieve the necessary information. We can update our objectives as follows:

| Objective  | Completed   |
|--|---|
| <b>Gathering the right sources where to obtain the necessary data. This data contains information concerning:</b> <ul style="list-style-type: none"><li>• <b>Past 24 hour weather statistics</b></li><li>• <b>Current weather status</b></li><li>• <b>Weather predictions</b></li><li>• <b>Pollen status</b></li></ul> |  |
| <b>Downloading this data to the user's device in order to be able to let the application handle this data.</b>   |   |
| <b>Read the data collected from the internet and contain it within the application.</b>  |   |
| <b>Present this data to the user in an easy to read manner.</b>  |   |

The next step is to make our application download this information.

## 1.2 Download data

This part will cover the downloading of the files needed in order to access the data required. In addition, it is very important to know when the downloads are finished before we are able to show the information to the user.

### 1.2.1 Connection

Before we can actually start downloading, we have to check if there is a valid internet connection on the user's device. If so, we can go on to the next step and actually download the requested files. (5)

If there is no valid internet connection, the program will use previously downloaded data if these are available. If there is no previous data available, the application won't be able to reach its goal of showing the user actual weather information.

### 1.2.2 DownloadManager

"The download manager is a system service that handles long-running HTTP downloads. Clients may request that a URI be downloaded to a particular destination file. The download manager will conduct the download in the background, taking care of HTTP interactions and retrying downloads after failures or across connectivity changes and system reboots." (6)

With the manager concerning all the network operations required for downloading a file, we can fill in a request for each file we want the manager to download.

With this request, we can specify some information. Here are some common examples:

- Network type: define which connections can be used in order to download the requested file (wifi or/and mobile connection).
- Description and title: define the information shown in the notifications while the file is being downloaded.
- Destination: where should the file be placed once the download has finished.
- Notifications visibility: define if you want to show a notification at start, during and after the download.

In my case, I'm not using all of these examples. I've decided not to show any notifications at the start or during the download because the files are very small and I don't want to overload the user with additional unnecessary information. With the files being very small I've neither chosen to set a limit on the network type, any internet connection is valid.

What I have specified is the destination, this is very important because you always want to know where to find your files once they are downloaded. The files in this application are saved within the application's external files. This way the user doesn't regularly encounter these files on their device but they can find it if necessary.

Before actually downloading the file, we look if there are previous files like this one available on the device. If so, we delete them in order not to stack up too much place on the device.

At the end, we can finally enqueue the file to the manager, who will decide when the device is ready to download. Each of these enqueues are given a unique id which we need in the next step.

### 1.2.3 BroadcastReceiver



In order to decide whether the download has finished downloading we need to listen to some broadcasts being sent from the manager. There are constantly broadcast sent in the device by different applications or by the system itself. (7)

The manager itself has 2 different broadcasts as well:

- ACTION\_DOWNLOAD\_COMPLETE: sent when a download completes.
- ACTION\_NOTIFICATION\_CLICKED: sent when the user clicks on a running download either from a system notification or from the downloads UI.

We're not using any notifications so we have no need for the second broadcast. We do need the first broadcast because we can't start reading the data and filling our screen before the data is actually downloaded. Very important in this description is the 'a', when we have multiple files to download each of these downloads will send a broadcast when finished. In order to know which download the broadcast is from we use the enqueue id we talked about earlier. Besides from debugging issues it's not very important to know which specific file has finished downloading and which hasn't. More importantly we need to be sure that all files are finished before we start reading the data. In order to check this, whenever a download has finished it increments a counter. We know in this application we need three different files so when the counter reaches 2 the next finished download broadcast can trigger the start of the data reading.

We do now have all of the data downloaded to the device so we can update our objectives as follows:

| Objective  | Completed   |
|--|---|
| <b>Gathering the right sources where to obtain the necessary data. This data contains information concerning:</b> <ul style="list-style-type: none"><li>• <b>Past 24 hour weather statistics</b></li><li>• <b>Current weather status</b></li><li>• <b>Weather predictions</b></li><li>• <b>Pollen status</b></li></ul> |  |
| <b>Downloading this data to the user's device in order to be able to let the application handle this data.</b>   |  |
| <b>Read the data collected from the internet and contain it within the application.</b>  |   |
| <b>Present this data to the user in an easy to read manner.</b>  |   |

The next step is to make our application read or parse the data.

## 1.3 Read data

Once we've downloaded the necessary files we can start reading, or using the right terminology, parsing these files. We have two types of files, one is CSV the other one is XML. For both these layouts we need a separate parser. We use these to read out the files containing information about past weather statistics, current weather, predictions and pollen. All of this information is put into neatly organized classes so we can easily access it in the next chapter.

### 1.3.1 CSVReader

"In computing, a comma-separated values (CSV) file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format." (8)

In order to parse CSV files, I went looking online for a solution. After some research, most of the problems I've encountered were solved with the help of a library called 'opencsv'. Looking further into this library, I came to the conclusion that it was easy to use and exactly what I was looking for.

"Opencsv is a very simple CSV (comma-separated values) parser library for Java. It was developed because all of current CSV parsers I've come across don't have commercial-friendly licenses." (9)

This library contains a class called 'CSVReader', which is essentially a CSV-parser. With this class's constructor, we can give along some information:

- Filereader with specific file: we define a reader along with the CSV-file to be read.
- Separator: the default separator in a CSV-file is the comma ','. Although we can define another separator if necessary.
- Quote character: the default quote character in a CSV-file is the double quote '"'. Although we can define another separator if necessary.
- Skip lines: define the number of lines to be skipped before actually parsing the file.

In my case, we define the correct file the download manager saved for us earlier. The separator and quote characters are the default characters so no need to change anything there. But, at the start of our CSV there is some information we don't need and cannot accept as values otherwise our system would crash. Therefore, we skip the first 4 lines of the CSV file in order not to run into trouble reading the data and in order to collect the correct information.

The information derived from the CSV files is information concerning the current state and the past 24 hours. Values such as temperature, humidity, rainfall, air pressure and wind. This information is put into a class called the 'Weather' class. This class contains all of the information and can be called upon when we need it.

Because the information we derive from this CSV contains values for each of the past 24 hours we immediately use this information after one of the lines is read. In this case we use it to build up a graph (more on this in the next chapter).

### 1.3.2 XmlPullParser

The second type of file we want to parse is an XML-file.

"In computing, Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable." (10)

In order to do this we use an Android interface called 'XmlPullParser', this interface intercepts different kinds of events occurring while reading the xml. These events are:

- START\_TAG: an XML start tag was read "<...>".
- TEXT: text content was read, a certain string.
- END\_TAG: an XML end tag was read "</...>".
- END\_DOCUMENT: no more events are available.

In order to explain how does goes to work, I'm using an example I found online. (11)

The figure is a slide titled "How XmlPullParser works". It is divided into two columns: "XML File" and "Java code".

**XML File:**

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book ISBN="ISBN-001">
    <name>How To Be A Cat</name>
    <author>Tom</author>
    <price>$29</price>
  </book>
  <book ISBN="ISBN-002">
    <name>How To Be A Mouse</name>
    <author>Jerry</author>
    <price>$25</price>
  </book>
</books>
```

**Java code:**

```
XmlPullParser.setInput(new FileInputStream("src/Books.xml"),
    "utf-8");
int eventType = xmlPullParser.getEventType();
while (eventType != XmlPullParser.END_DOCUMENT) {
    switch (eventType) {
        case XmlPullParser.START_TAG:
            if ("book".equals(xmlPullParser.getName())) {
                String isbn = xmlPullParser.getAttributeValue(
                    XmlPullParser.NO_NAMESPACE, "ISBN");
            }
            if ("name".equals(xmlPullParser.getName())) {
                String name = xmlPullParser.nextText();
            }
            break;
        case XmlPullParser.END_TAG:
            if ("book".equals(xmlPullParser.getName())) {
                // ...
            }
            break;
        default:
            break;
    } // end switch
    eventType = xmlPullParser.next();
} // end whiles
```

eventType:

Figure 4: XmlPullParser explanation

The first event we use is the END\_DOCUMENT, we use this to decide when our document is fully read and we can exit our while-loop. As long as this event does not occur, we keep reading the xml file.

Inside the while-loop we have a switch-case statement to decide which event has occurred. The reader continues to read the document and looks for any events to occur.

The first event who occurs is a `START_TAG`, of course we're not interested in all the tags inside of the XML. So, we decide what to do when a `START_TAG` occurs by different if-statements. In this case, we are interested in the tags with values `'book'` and `'name'`. So, we declare an if-statement where we check if the name who goes along with this `START_TAG` is equal to `"book"` and another if-statement where to name goes along with `"name"`. If not, we just skip the line as is done with the first two lines inside this example XML.

The third `START_TAG` we encounter has the value `'book'`. In this tag we are interested in its attribute `'ISBN'` so we save this into a String called `isbn` using the function `getAttributeValue()`. This function requires two parameters: the first one being a namespace, if there is one, and the second one being the name of the attribute we're looking for. This XML does not use any namespaces so we pass along the `NO_NAMESPACE` value, the name of the attribute we need is `"ISBN"`. The value of the String `isbn` is now equal to `"ISBN-001"`.

The next tag we are interested in has the value `'name'`. We declare another if-statement which looks for a `START_TAG` with the value `'name'`. We encounter it on the 4<sup>th</sup> line of the XML file. Now we need the text next to this tag so, we save this text into a String called `name` using the function `nextText()`. The value of the String `name` is now equal to `"How To Be A Cat"`.




At the end of line 4 we encounter an `END_TAG` with the value `'name'`, we have no interest in this tag so we move on.

The next interesting event occurs at line 7 where we encounter an `END_TAG` with the value `'book'`. In human language, this means we have finished reading all the values of one certain book. It is not described in this example but we could save the variables we've noted like the strings `'isbn'` and `'name'` into a different class so we don't lose this information when we overwrite these variables with information about the next book.

After we've checked all possible tags inside the switch-case statement or did not encounter any interesting events, we call `next()` to go to the next parsing event. When we reach the end of the document the `END_DOCUMENT` tag becomes the current `eventType` and we quit the while loop, the document has been fully read.

In the case of the project concerning this book, all of the necessary information is saved into weather classes so we can use this information for displaying the information on the screen.

We do now have all the data from the downloaded files read and organized into classes so it is contained into the application. We can update our objective list as follows:

| Objective  | Completed   |
|--|---|
| <p><b>Gathering the right sources where to obtain the necessary data. This data contains information concerning:</b></p> <ul style="list-style-type: none"> <li>• <b>Past 24 hour weather statistics</b></li> <li>• <b>Current weather status</b></li> <li>• <b>Weather predictions</b></li> <li>• <b>Pollen status</b></li> </ul> |  |
| <p><b>Downloading this data to the user's device in order to be able to let the application handle this data.</b></p>  |  |
| <p><b>Read the data collected from the internet and contain it within the application.</b></p>   |  |
| <p><b>Present this data to the user in an easy to read manner.</b></p>   |   |

The next step is the most important one, use this data and finally present it on the user's device.



## 1.4 Show data

Now that we have all the data at our disposal, we only need to show it in an easy-to-read manner to the user.

The idea I've come up with consists of 7 different pages through which you can swipe. The first one containing information for today, the second one for tomorrow and so on. In order to do that I needed a couple of classes and methods.

### 1.4.1 Swipe pages

I wanted to have the application's swiping go smoothly and not needing any loading time. So we could only use one activity for these 7 pages because loading an activity takes some time, although only milliseconds it doesn't look smooth. (12) (13) (14)

First we need the activity. "The Activity class serves as the entry point for an app's interaction with the user, providing the window in which the app draws its UI." (15) In other words an activity is the base of what your application does. Linked to this activity we have a layout file which determines what the user gets to see on the screen. We make use of one base on which we build our 7 pages.

Each page is a fragment. A fragment is a portion of a user interface, in one activity you can have multiple fragments. You can arrange them neatly on one screen or you can specify that the fragment uses the whole screen.

We use a fragment for each page so 7 fragments and manage them using a ViewPager.

The viewPager is a layout manager which allows the user to swipe left and right through pages of data. In order to generate all these pages we need a PagerAdapter in our case we use the FragmentStatePagerAdapter.

### 1.4.2 TextView, ImageView

TextViews and ImageView are widgets from Android itself. TextView which displays text to the user and optionally allows them to edit. ImageView shows an image instead of text.

These views are easily added to the layout in the graphical design or straight into the layout XML. The text and images can be logically set or fixed from the start.

They can also be made clickable so when the user pushes them a certain action can be programmed.

### 1.4.3 PopupWindow, Radio buttons

The PopupWindow class speaks for itself. We can use this class in order to create a popup window. It's a floating container that appears on top of the current activity. (16) (17)

Radio buttons allow the user to select one option from a set. Only one option can be chosen each time. We'll use both if these for the selection of the location.

#### 1.4.4 ScrollView

Sometimes one screen is not enough to show all of the information you want to show. Therefore we can add a scrollView which enables the user to scroll down, up, left or right depending on which settings you apply to this view. With this we can add more information in just one activity where the user can scroll through.

#### 1.4.5 GraphView

To give a clear view about the course of the past 24 hours concerning temperature, humidity and rainfall I'm using a library called 'GraphView'.

"GraphView is a library for Android to programmatically create flexible and nice looking diagrams. It is easy to understand, to integrate and to customize." (18)

GraphView is inserted through the XML layout file as seen in this example.

```
<com.jjoe64.graphview.GraphView
    android:layout_width="match_parent"
    android:layout_height="200dip"
    android:id="@+id/graph" />
```

Figure 5: GraphView XML layout

In order to get data into this graph you need a set of DataPoints, which are then inserted into the graph. In the next example, we see a line graph being generated through Java code.

```
GraphView graph = (GraphView) findViewById(R.id.graph);
LineGraphSeries<DataPoint> series = new LineGraphSeries<>(new DataPoint[] {
    new DataPoint(0, 1),
    new DataPoint(1, 5),
    new DataPoint(2, 3)
});
graph.addSeries(series);
```

Figure 6: GraphView code snippet

When this code is executed, we get something like the following.

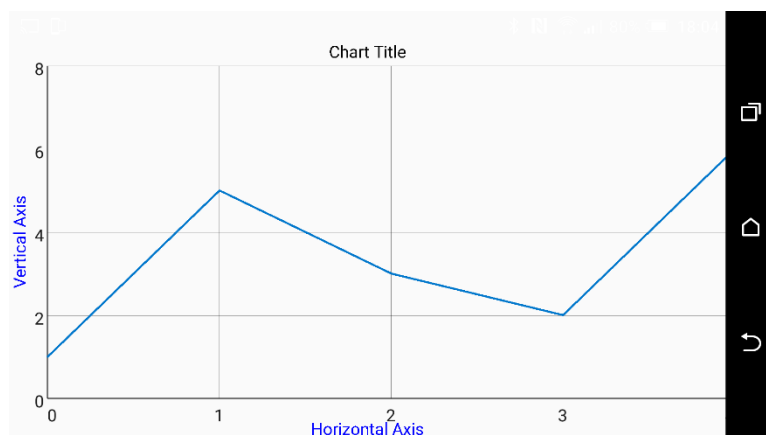


Figure 7: GraphView example

## 2 Logic's implementation

In this chapter we will revisit all of the information from chapter 1 but now I will show you how I've implemented this logic into the project.

But, before we start I have to explain the application's `AndroidManifest.xml` or the manifest file.

The figures used in this chapter are from the applications' code or screenshots from the running application.

### Manifest

What is the manifest file and what does it do?

"Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- [...]
- It declares the permissions that the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.
- [...]
- It declares the minimum level of the Android API that the application requires.
- [...]" (19)

Based on this information we will need to add some things to the manifest file during our progress in order for everything to work. Once we've reached to point, these things will be named and referred back to the manifest.

## 2.1 Sources

We've gathered the correct sources so now we can implement them into the application. The application covers different locations and for each of these locations the URL is different. So based on which location the user has selected we decide which URL's to use.

### Flowchart

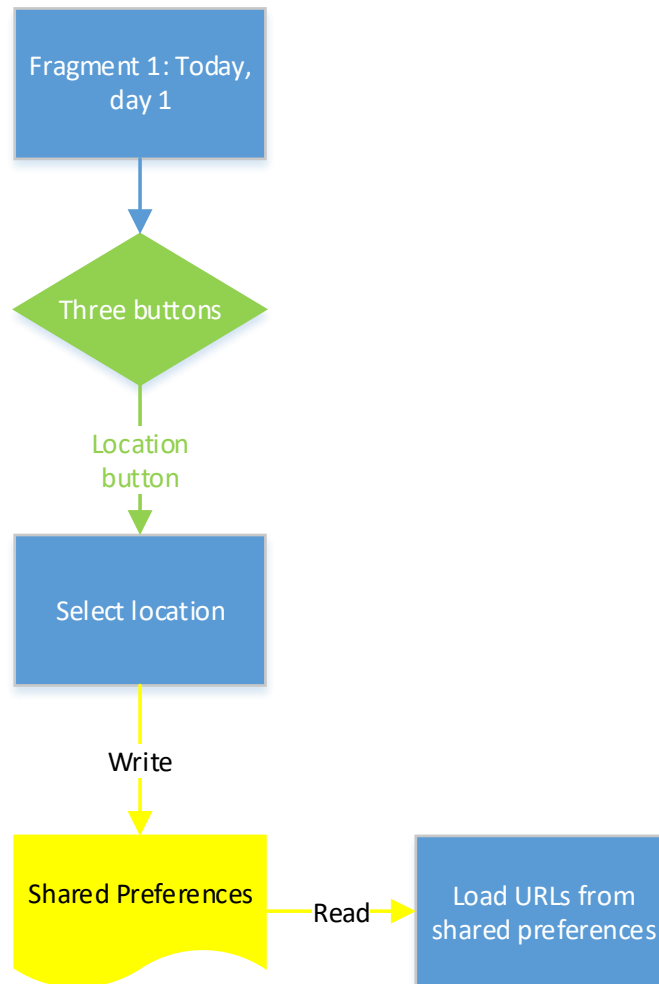


Figure 8: Sources flowchart

## Selecting location

The user has the ability to change his preferred location. In this application we have 13 locations available. In the first screen the user gets to see, there's the possibility to change the preferred location. This location is then stored into the shared preferences of the application.

This SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write these pairs. When the application is closed and rebooted, these values are still present so the previous preferred location is still the active location.



Figure 9: Edit location screenshots

In order to change the location the user presses the highlighted location image in the top left corner. The user is then provided with the 13 different locations from which they can choose.

Once the user selects a certain location, there is a toast message at the bottom confirming the selected location, the shared preferences are modified and the activity with all its 7 pages is recreated implementing the data corresponding the selected location.

In the next figure you can see the code required in order for the application to edit the shared preferences and recreate the activity. It shows 2 of the 13 different possibilities corresponding to the 13 different locations.

```
case R.id.avilaBtn:
    popupWindow.dismiss();
    Toast.makeText(getActivity().getApplicationContext(),
        "Location: Ávila", Toast.LENGTH_SHORT).show();
    editor.putString("csvURL", Constant.AVILA_URLS[0]);
    editor.putString("xmlURL", Constant.AVILA_URLS[1]);
    editor.putString("location", Constant.AVILA_URLS[2]);
    editor.putString("pollenLocation", Constant.AVILA_URLS[3]);
    editor.commit();
    getActivity().recreate();
    break;
```

*Figure 10: Edit location snippet*

So, for each different option selected to following steps are performed:

- Edit the URL for the CSV file in the shared preferences.
- Edit the URL for the XML file in the shared preferences.
- Edit the location to be put on the screen in the shared preferences.
- Edit the location used to compare strings in the shared preferences.
- Commit these changes.
- Recreate entire activity.

We now have to correct location and its corresponding URLs saved into the shared preferences. We will now see where these URLs are saved in the application. (20)

## URLs

As you may have noticed in the previous figure, we fetch our URL values from an interface called Constant. In this interface we've hardcoded the URLs with their responding location. The positive thing is that the URLs are nicely ordered and are easily accessible from anywhere in the application. The values are saved as strings in a string array, one for each location.



Figure 11: Constant UML

The next figure shows an example of such a string array.

```
String[] AVILA_URLS = {
    "http://www.aemet.es/es/eltiempo/observacion/ultimosdatos_2444_datos-
    horarios.csv?k=cle&l=2444&datos=det&w=0&f=temperatura&x=h24",
    "http://www.aemet.es/xml/municipios/localidad_05019.xml",
    "Ávila",
    "AVILA"
};
```

We have saved the name in two different forms, the first one is shown on the screen thus the accents and capital letters are correct. While the second one is used in order to compare strings when there are no accents used as is the case in the pollen XML.

We do now have access to the URLs; do not forget that when we want to access the correct URLs we have to use the ones stored in the shared preferences in order to comply to the users' location preference.

We can now start downloading the files corresponding to these URLs.

## 2.2 Downloading

The downloading happens automatically when the application is booted, so all of this is done in the MainActivity. This activity is also responsible for generating the ViewPager which generates our 7 swipeable pages, but this will be explained in the next chapter.

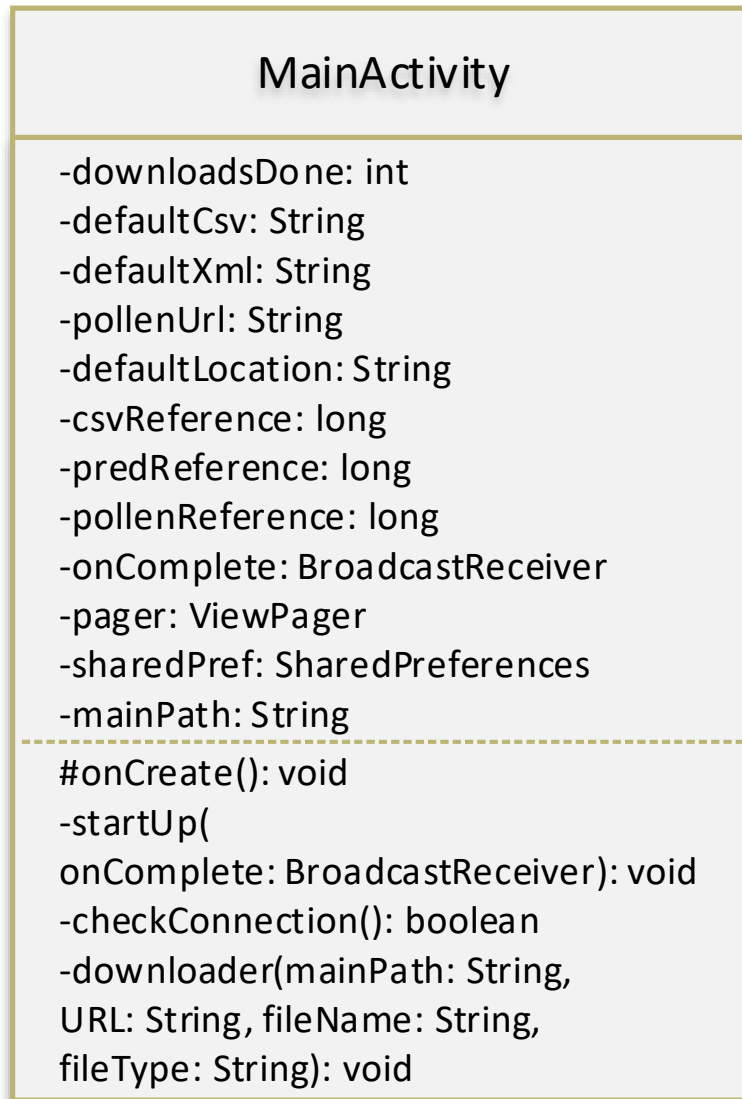


Figure 12: MainActivity UML

In order to be able to download files from the internet, the application needs the permission to use an internet connection. Therefore, we need to add another permission into the manifest.

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



Flowchart

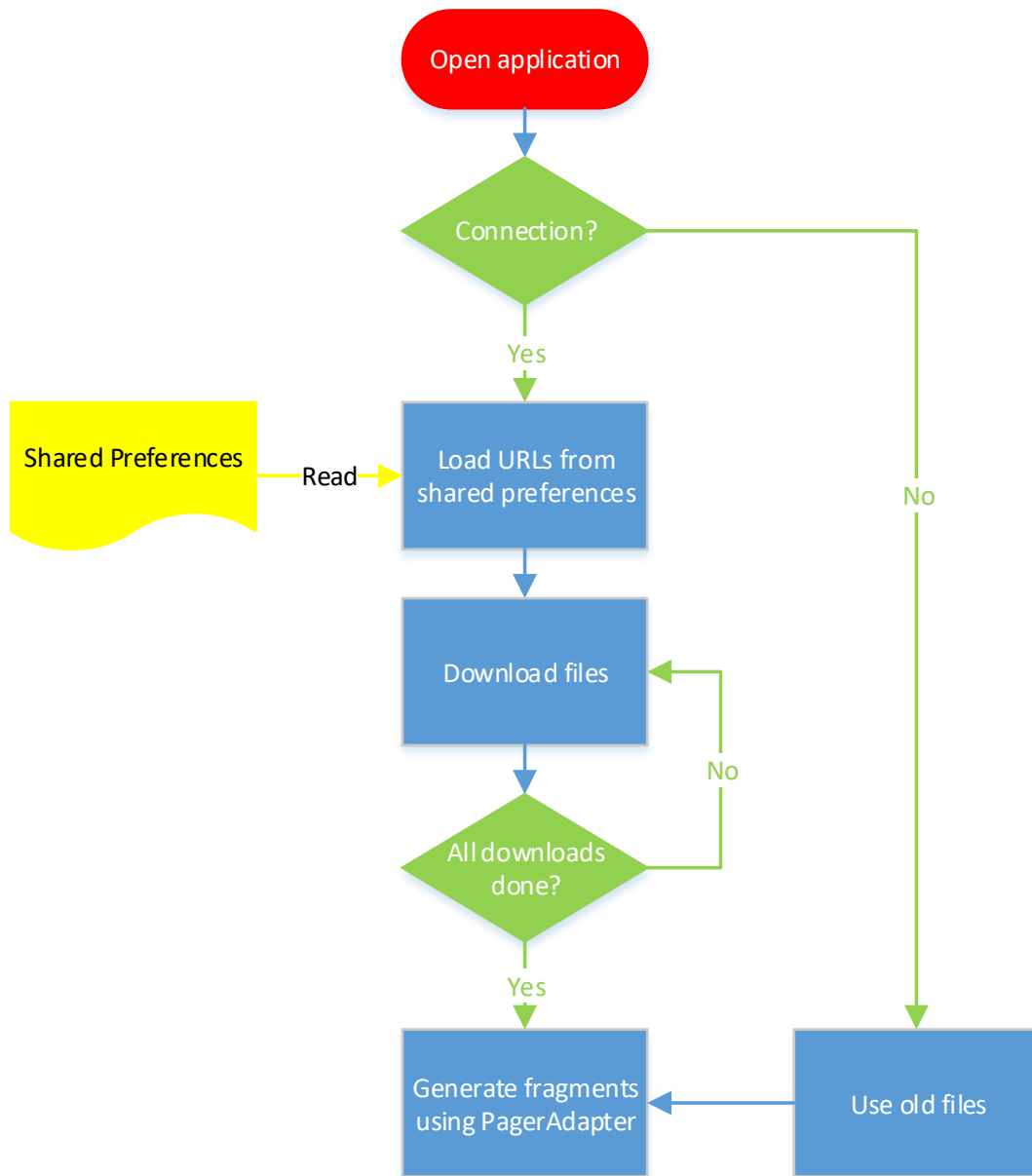


Figure 13: Downloading flowchart

### 2.2.1 Connection

Before downloading, we have to check if there is a valid internet connection.

The method `checkConnection` returns a Boolean, either true or false depending on the fact that there is a connection or not. This Boolean is then used to depend if we can start downloading or not.

In order to do so we use the class `ConnectivityManager`; this is a class that answers queries about the state of network connectivity. From this class we use the function `getActiveNetworkInfo()` which returns the currently active default data network. If there is none, it returns null.

Therefore, we check whether this function returns null. If not, it means there's currently an active internet connection so we return true and are able to start downloading.

In order for the application to access information about networks, we need to add a permission into the manifest.

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

*Figure 14: Network state manifest*

We are now able to check for a valid connectivity to the internet.

### 2.2.2 DownloadManager

So, first we check whether there is an internet connection available. If not, a message pops up notifying the user there is no valid internet connection and we use the known data. The adapter is added to the view pager and the fragments are produced which we will explain later on.

If we do have a valid internet connection, the `checkConnection()` method returns true as seen before. We decide which URLs to use based on the URLs stored in the shared preferences. This also requires a default value in case the required value is not found, we've chosen to add the URLs for Valladolid.

```
sharedPref =  
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());  
  
String csvURL = sharedPref.getString("csvURL", defaultCsv);  
String xmlURL = sharedPref.getString("xmlURL", defaultXml);
```

*Figure 15: Access Shared Preferences snippet*

These URLs can now be used in order to download the correct information related to the users' requested location.

Next, we call the downloader method. We call this method 3 times, once for each file to be downloaded.

We pass along some arguments with this function.

- **mainPath**: this is the path to the application's external files directory where we will save our downloaded files. We use this to determine the download destination. This has been declared as follows.

```
final String mainPath = getExternalFilesDir(null) + "/";
```

*Figure 16: File location snippet*

- **URL**: previously defined URL who leads to the file you want to download.
- **fileName**: define a name for the file.
- **fileType**: describe the type of the file, we need this in order to give the download a unique id so we can then check when the download is finished.

As noted before we first build up a request to pass along to the manager. This request contains the URL. With this request, we can specify some properties but with the choice of not showing any download notifications, we only need to adjust the notification visibility and set it to hidden.

We then declare a file with the mainPath and fileName. This is in fact the file we are downloading, but we need to declare it before the download to check if the file does not already exist. If so, we delete the file before downloading it again in order to save up space on the device.

We set the destination for the file, as discussed before we save the files into the applications external files directory.

At last, we can enqueue the download to the manager. While doing this, each enqueue generates a unique id, we need this id for the BroadcastReceiver in order to know which download was finished. So, we save the id based on which fileType we are downloading. We have 3 different references: csvReference for the CSV file containing the historic and current information, predReference for the XML file containing the predictions and pollenReference for the XML file containing information about the pollen.

### 2.2.3 BroadcastReceiver

We need to check whether the downloads are finished in order to start filling the screen.

In order to use a receiver, you need to register it. With this registration we declare the name of the receiver as well what kind of broadcasts this receiver's interests are, in this case we want to know when a download has completed. Therefore, we use the `ACTION_DOWNLOAD_COMPLETE` constant which checks broadcasts concerning downloads which have been completed.

We declare the BroadcastReceiver with the name `onComplete`. Every time this receiver receives a broadcast, this broadcast comes along with a unique id which is linked to the id given by the downloader. We then compare this id with the id's we've filled in earlier to know which download has completed.

Once we receive a broadcast with an id we recognize, we call the `startUp` method. This method checks the `downloadsDone` variable, this variable is in fact a counter. For each download who has been completed this counter goes up, once we reach 3 which means all of the necessary files are done downloading the receiver is unregistered since we don't need it anymore and an adapter is set for the ViewPager. This causes the creation of our 7 pages which we'll cover in the next chapter.

The files are now downloaded to the device and available for reading.

## 2.3 Read data

We can now start reading the downloaded data which was saved on the device. As mentioned before in order to contain this data and be able to use it throughout the application we save it inside Weather classes and Pollen classes.

### Weather class

The weather class will contain our information about the weather statistics and make it easy for us to obtain it wherever necessary. The next figure shows a shortened version of the UML from this class. I've shortened the method's because they are all just getters and setters for each attribute in order to be able edit and read them.



Figure 17: Weather UML

As you can see, it has the ability to hold a lot of information. Although we don't always need all of this information.

Concerning the CSV file where we find information about the current weather situation and the past 24 hours we only need certain values:

- temperature
- windSpeedCsv
- windDirectionCsv
- precipitation
- humidity
- airPressure

These are the values we can derive from the CSV file. We are able to fill up to 24 of these classes, one for each passed hour including the current hour.

The other values are used when reading the XML file where we have information concerning weather predictions which are available per six hours.

## Pollen class

The Pollen class will contain our information about the pollen statistics and make it easy for us to obtain it wherever necessary. The next figure shows a shortened version of the UML from this class. I've shortened the method's because they are all just getters and setters for each attribute in order to be able edit and read them.

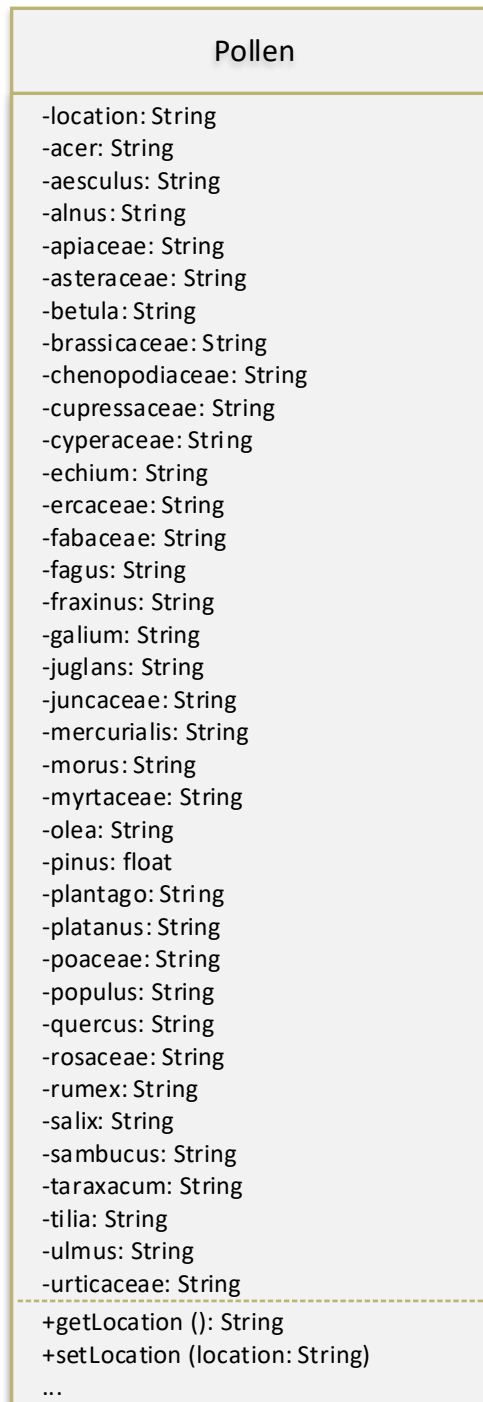


Figure 18: Pollen UML

As you can see, there are a lot of different types of pollen for which we can acquire information.

Each class concerns one location; we have 13 locations in total so in the end there will be 13 objects of this class.

Although not every location contains all of these different types of pollen. Some locations don't supply information about a certain type of pollen because they do not measure it or simply because this type is not common in that area.

This data is updated on the website once every week with data for the current status and a prediction for the whole week. We only make use of the prediction to supply a clear view for the whole week to come.

### 2.3.1 CSV reader

Before explaining how we derive the data from the CSV file, let me first give you an example of what one of these CSV files might look like. The next figure shows a part of such a file.

"Valladolid"

Actualizado: martes, 14 marzo 2017 a las 12:22 hora oficial

"Fecha y hora oficial","Temperatura (°C)","Velocidad del viento (km/h)","Dirección del viento","Racha (km/h)","Dirección de racha","Precipitación (mm)","Presión (hPa)","Tendencia (hPa)","Humedad (%)"

"14/03/2017 12:00","13.4","13","Nordeste","37","Nordeste","0.0","942.3","0.3","56"

"14/03/2017 11:00","11.9","9","Nordeste","39","Nordeste","0.0","941.9","0.1","61"

"14/03/2017 10:00","10.0","9","Nordeste","35","Norte","0.0","942.0","-0.3","69"

"14/03/2017 09:00","7.7","10","Nordeste","33","Nordeste","0.0","942.0","0.3","76"

"14/03/2017 08:00","6.2","10","Nordeste","33","Norte","0.0","941.8","0.5","81"

...

*Figure 19: CSV file example*

As we've mentioned before, the first 4 lines ("Fecha hora... is one line) are information we don't need so we can safely skip these 4 lines.

Starting from the fifth line we can see the different values each surrounded by "" and separated by a comma. We have date and hour, temperature, windspeed, wind direction, gust speed, gust direction, rainfall, air pressure, air pressure trend and air humidity. Each of these for each of the past 24 hours.

Now that we know what we want to read, we can start reading. And saving the information into a Weather class(es).



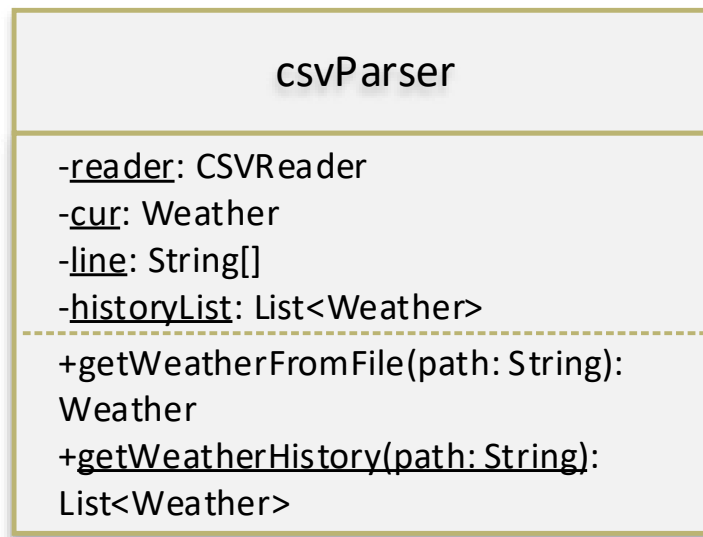


Figure 20: csvParser UML

First, we collect information about the current status.

In order to put the CSV data into a Weather class I've made a csvParser class with a method called getWeatherFromFile which returns a Weather object containing the data.

With this function we pass along the location of where the file is supposed to be. There is some safety built in to cover that by any chance the file is not there using some exception handlers.

We start by initiating a CSVReader, which reads a stream of data coming from the CSV-file. This stream contains each of the characters in the file one by one. We also initiate a Weather object in which we will store our data.

We then specify the reader using the stream of data, which separator to look for (here it's just the default comma), which quote character to look for (also default) and the 4 decides we skip the first 4 lines.

Each line is now transformed to a String array with each element being a value. We can then fill our Weather object using the corresponding value in this String array. At the end, we return the Weather object which can then be used to obtain this information.

On the other hand, we also need information about the past 24 hours. We use the getWeatherHistory method which works more or less the same as the previous method with the only difference being it returns a list of Weather objects for each hour instead of just one.

### 2.3.2 Xmlparser

Next up is the parsing of the XML files. The next figure gives you an example of what an XML file containing information about the weather predictions might look like. Note that this is just a small part of the XML as most of the tags are collapsed and most of the information you see is available for each of the following 7 days.

```
▼ <prediccion>
  ▼ <dia fecha="2017-04-24">
    <prob_precipitacion periodo="00-24"/>
    <prob_precipitacion periodo="00-12"/>
    <prob_precipitacion periodo="12-24">60</prob_precipitacion>
    <prob_precipitacion periodo="00-06"/>
    <prob_precipitacion periodo="06-12">0</prob_precipitacion>
    <prob_precipitacion periodo="12-18">50</prob_precipitacion>
    <prob_precipitacion periodo="18-24">20</prob_precipitacion>
    <cota_nieve_prov periodo="00-24"/>
    <cota_nieve_prov periodo="00-12"/>
    <cota_nieve_prov periodo="12-24"/>
    <cota_nieve_prov periodo="00-06"/>
    <cota_nieve_prov periodo="06-12"/>
    <cota_nieve_prov periodo="12-18"/>
    <cota_nieve_prov periodo="18-24"/>
    <estado_cielo periodo="00-24" descripcion=""/>
    <estado_cielo periodo="00-12" descripcion=""/>
    <estado_cielo periodo="12-24" descripcion="Intervalos nubosos con lluvia escasa">43</estado_cielo>
    <estado_cielo periodo="00-06" descripcion=""/>
    <estado_cielo periodo="06-12" descripcion="Despejado">11</estado_cielo>
    <estado_cielo periodo="12-18" descripcion="Intervalos nubosos con lluvia escasa">43</estado_cielo>
    <estado_cielo periodo="18-24" descripcion="Intervalos nubosos">13</estado_cielo>
    ▶ <viento periodo="00-24">...</viento>
    ▶ <viento periodo="00-12">...</viento>
    ▼ <viento periodo="12-24">
      <direccion>S0</direccion>
      <velocidad>10</velocidad>
    </viento>
    ▶ <viento periodo="00-06">...</viento>
    ▶ <viento periodo="06-12">...</viento>
    ▶ <viento periodo="12-18">...</viento>
    ▶ <viento periodo="18-24">...</viento>
    <racha_max periodo="00-24"/>
    <racha_max periodo="00-12"/>
    <racha_max periodo="12-24"/>
    <racha_max periodo="00-06"/>
    <racha_max periodo="06-12"/>
    <racha_max periodo="12-18"/>
    <racha_max periodo="18-24"/>
    ▼ <temperatura>
      <maxima>26</maxima>
      <minima>8</minima>
      <dato hora="06">9</dato>
      <dato hora="12">25</dato>
      <dato hora="18">22</dato>
      <dato hora="24">15</dato>
    </temperatura>
    ▶ <sens_termica>...</sens_termica>
    ▶ <humedad_relativa>...</humedad_relativa>
    <uv_max>7</uv_max>
  </dia>
  ▶ <dia fecha="2017-04-25">...</dia>
```

Figure 21: XML file example



Figure 22: xmlParser UML

Our XMLParser class has a function called `getWeatherListFromFile` which returns a list of Weather objects, one for each day. We fill each of these objects with information from a corresponding day. At the end we return a list of 7 Weather objects.

We start by determining some tags we'll need and applying them to a constant. Each tag must correspond to a tag we're interested in from the XML file.

```

private static final String KEY_DIA = "dia";
private static final String KEY_PROBPRECI = "prob_precipitacion";
...
  
```

Figure 23: XML tags snippet

We start by creating a XmlPullParser and give it the correct input, a stream containing the information from the file. This parser will now listen to this stream and generate different event types based on which tag it encounters.

The first one we see is KEY\_DIA, this is where information from a new day starts so we initiate a Weather object.

Next, we have KEY\_PROBECI which contains information about the chance of rain, but we have this information for different hour intervals during this day. So, we need to check which hour each tag belongs to. The next figure is a part of the code for the hour intervals 00-24 and 00-06. (21)

```
else if (tagname.equalsIgnoreCase(KEY_PROBPRECI)) {  
  
    //Because with some there is no information yet.  
    if (xpp.getAttributeCount() > 0) {  
        if (xpp.getAttributeValue(null, "periodo") != null &&  
xpp.getAttributeValue(null, "periodo").equals("00-24")) {  
            curWeather.setRainProbDay(xpp.nextText());  
        }  
  
        if (xpp.getAttributeValue(null, "periodo") != null &&  
xpp.getAttributeValue(null, "periodo").equals("00-06")) {  
            curWeather.setRainProb0006(xpp.nextText());  
        }  
    }  
    ...  
}
```

Figure 24: KEY\_PROBPRECI snippet

We first check whether there is an attribute with the tag because for days further in the future there isn't as much information available yet as for the day in the example.

We then check to which interval the tag belongs and add it to the corresponding variable in the Weather object of the current day.

To determine the state of the sky, determined by KEY\_CIELO we use the same method. You can see that these tags have more than one attribute but we are only interested in the first one stating the hour intervals. The second one gives a String with information about the status of the sky but we use the integer accompanied by it.

With the last two tags, in order to find the predictions concerning temperature and wind, we approach a different method. The information is in fact hidden inside of these tags, so we first need to get inside and then start another loop, which runs through the information inside of the first tag.

Inside this second loop, we find information about the predicted maximum temperature, minimum temperature and temperatures for different hours of the day. Some of these were left out of this example in order not to overwhelm the snippet.

When we reach the end tag KEY\_TEMP it means we have read all the information available and finish the loop.

A small variant of this logic is used in order to obtain wind information.

## 2.4 Show data

Now that we have all the data available inside Weather and Pollen classes we can start filling up the screen with this information. As noted before, we're making 7 pages we can swipe through, one for each day. These 7 pages are managed in one activity, next we have an activity for the pollen information and one for historic weather data.

The activity which contains the viewPager is yet again the MainActivity.

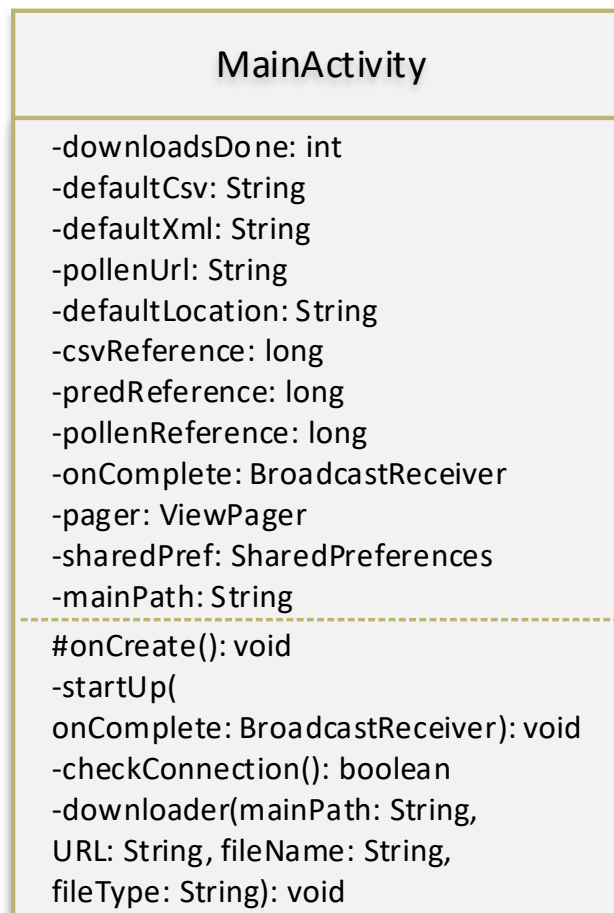


Figure 25: MainActivity UML

As mentioned before the process of creating the 7 fragments, managed by a viewPager's adapter, is triggered from the startUp method once all of the files have been downloaded.

Flowchart

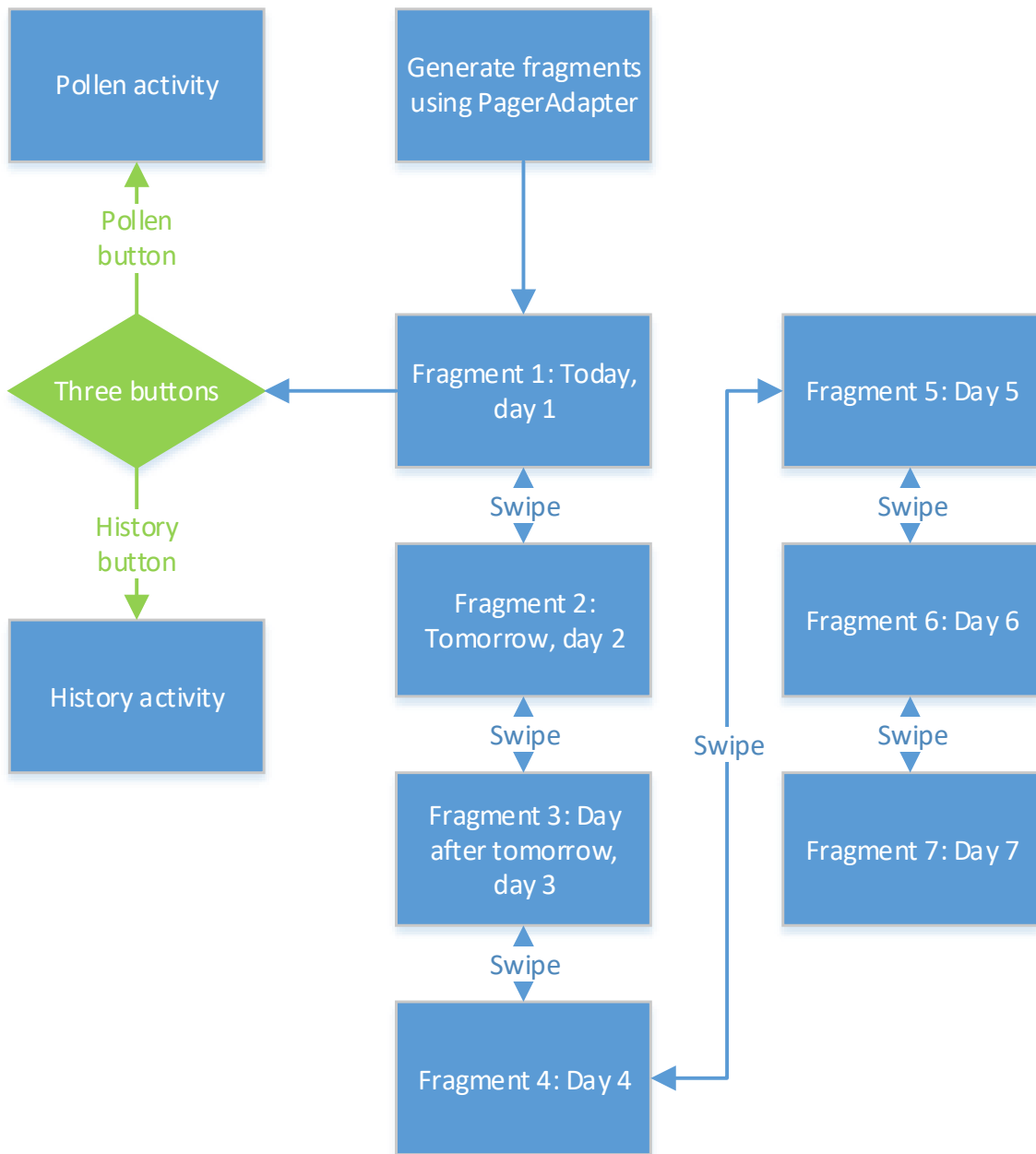


Figure 26: Data showing flowchart

### 2.4.1 Swipe pages

The layout from the main activity on which we build our 7 pages is solely filled with one viewPager. This pager manages the layouts of all these 7 pages. Well in fact we only use 2 different layouts for these 7 pages.

The logic behind these 7 pages is based upon 3 different classes where we decide what each layout element shows.

These views are being supplied to the viewPager by a PagerAdapter, more precisely we use a FragmentStatePagerAdapter which is easier for implementing fragments.

#### *PagerAdapter*

This adapter manages our 7 separate fragments. The creation of these fragments is as follows.

At first, we call upon our readers from the previous chapter which supply us with a weatherlist containing all the predictions and a csvWeather which contains the current weather data.

We then supply all of the possible pages, 7 in total. Each of these pages is an instance of one of the 3 different classes FirstFragment, SecondFragment or ThirdFragment. Along with these instances we pass the information necessary in order to fill up each page.

```
case 0: return FirstFragment.newInstance(weatherlist.get(0), csvWeather, location);
case 1: return SecondFragment.newInstance(weatherlist.get(1), location);
case 2: return ThirdFragment.newInstance(weatherlist.get(2), 2, location);
case 3: return ThirdFragment.newInstance(weatherlist.get(3), 3, location);
case 4: return ThirdFragment.newInstance(weatherlist.get(4), 4, location);
case 5: return ThirdFragment.newInstance(weatherlist.get(5), 5, location);
case 6: return ThirdFragment.newInstance(weatherlist.get(6), 6, location);
default: return ThirdFragment.newInstance(weatherlist.get(5), 5, location);
```

*Figure 27: Fragment instances snippet*

For example, case 0: from the weatherlist we'll need the first object which concerns today's predictions, we'll need the csvWeather which concerns the current state and we pass along the location specified in the shared preferences.

We now know how the pages are called upon; the next step is to see how each page is made. As said before there are 3 different fragment classes and 2 different layouts. We will discuss each unique combination; the others are merely supplied with different data.

### Fragment class 1, layout 1

The next figure shows an example of what this one might look like.

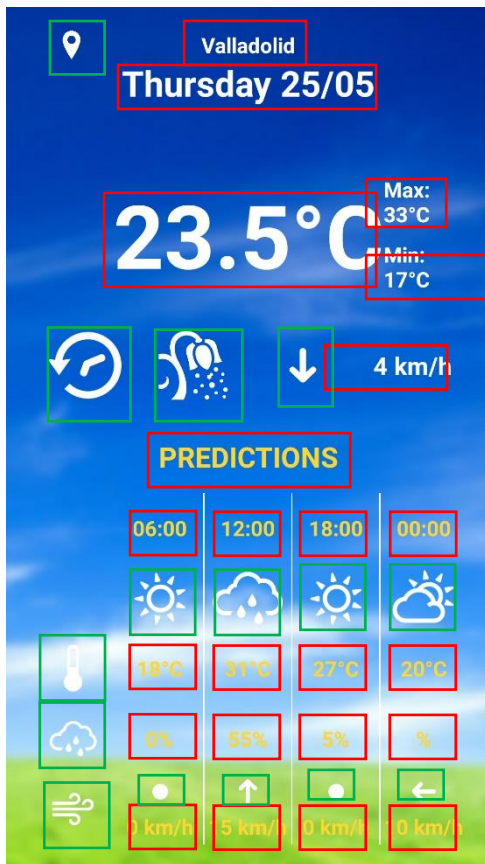


Figure 28: Fragment class 1, layout 1

As we can see, there's a lot going on. First of all, we can see that the background is filled with an image. This image is related to the current weather state. So, when it would be raining, there would be rain in the background.

Next, we have a lot of textViews, all of the red noted views are textViews. They are very important in giving information to the user. Some of them have fixed text, while others are based on the data we've assembled.

Then we have the green ones, these are imageViews, they are in fact just images on the screen. Also here some are fixed while others depend on the predictions.

The upper part consists of information obtained from the CSV file, that is, the current status. While the bottom half shows the predictions.



### Fragment class 2, layout 1

The next figure shows an example of what this one might look like.



Figure 29: Fragment class 2, layout 1

If the previous one was for today, then this one is information about tomorrow. As you can see, we've used the same layout.

But the big difference is that here the upper part also consists of predictions. So both, the upper and the bottom half, are based upon predictions. Same for the background, it looks like they're predicting some rain.

### Fragment class 3, layout 2

The next figure shows an example of what this one might look like.



Figure 30: Fragment class 3, layout 2

For the remaining fragments we use a different layout. This one contains less information for the simple reason that the data we download doesn't contain as much information for days as far ahead.

We've got predictions for the temperatures, the state of the sky (looks like it'll be sunny), the chance for rainfall and the wind.

Now we know what the different types of fragments look like, it's time to look at how we've managed to make them look like they do. Let's take a look at the logic behind each of the 3 types of fragment classes.

### *FirstFragment*

This one is responsible for solely the first day, so in fact the current day.



Figure 31: FirstFragment UML

We use a Calendar instance to acquire today's date and add it to the screen, and to decide the current hour in order to set the correct background image depending on the prediction for the current time.

The radioGroup is for when the user selects a location.

When the PageAdapter generates these fragments, the first method that is called is the newInstance. As seen before we pass along all the information necessary in order to fill the screen. Inside this method we make a Bundle object who concerns about the arguments of this fragment. We take all the information given by the PageAdapters' arguments and set them as arguments from this fragment.

```
args.putFloat("curTemp", csvWeather.getTemperature());
args.putInt("curSpeed", csvWeather.getWindSpeedCsv());
args.putString("curDirection", csvWeather.getWindDirectionCsv());
```

Figure 32: Set fragment arguments snippet

These are some examples of how to put the information in the Bundle object which describes the fragments' arguments. It's, like the shared preferences, with a key-value pair. For example, in the last shown line we put the given location

String into an argument with the name location. We do this for all of the necessary values.

When this method finishes it returns the fragment and the onCreateView method is called. This is where we produce the actual view and give all the layout elements their values.

The views must first be associated with a view in the layout file. Now each of these views can be modified as desired. The text views are filled with the correct data. This data is fetched from the arguments we formed when creating the instance. We can now call on them by using the getArguments() method. (22)

```
tempView.setText(getArguments().getFloat("curTemp") + "°C");
speedView.setText(getArguments().getInt("curSpeed") + " km/h");
directionImage.setImageDrawable(selectWindDir(
    getArguments().getString("curDirection")));
```

*Figure 33: Get fragment arguments snippet*

In order for the sky icons to show the correct icon linked to the weather state we made a method that returns the correct drawable based on the given sky state selectIcon.

Next, we have the wind icons which show the direction of the wind. We do the same as with the sky icons, we give the method selectWindDir a wind direction as a String and it returns the correct drawable.

At last, we decide the correct background for this fragment using the setBackground method. This does not only decide the background but sets it as well. But very important here was that not every user is using the same device, and not all the devices have the same screen size. Therefore the image has to be scaled correctly before adding as background.

```
Display display = getActivity().getWindowManager().getDefaultDisplay();
Point size = new Point();
display.getSize(size);

Bitmap bmp;

bmp = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(
    getResources(), R.drawable.clear_sky), size.x, size.y, true);

LinearLayout ly = (LinearLayout) v.findViewById(R.id.first_frag);
Drawable dr = new BitmapDrawable(getResources(), bmp);
(ly).setBackground(dr);
```

*Figure 34: Resize background image*

Therefore, we first determine which display is being used, along with that we can decide the size. We then select the correct image based on the sky state and we rescale it. This is only possible if the image has a bitmap or .bmp type. But we cannot set the background with a bitmap type so at the end we convert the converted bitmap, which now has the correct size along with the screen size, to a drawable which we then use to set the background. (23)

### *Second and third fragment classes*

The seconds' class only difference with the first fragment class is the data used to fill the layout elements. For tomorrow we can only use predictions, there is no current data available.

The thirds' class just has a lot fewer layout elements since there isn't a lot of information available from two days in the future.

## 2.4.2 ScrollView

The scrollView allows us to expand our layout on a single view. We are not limited to the boundaries of the screen itself. We can make a view which is bigger than the screen and scroll through it all on one screen.

In this project we use it to display a detailed list of different types of pollen and their status.

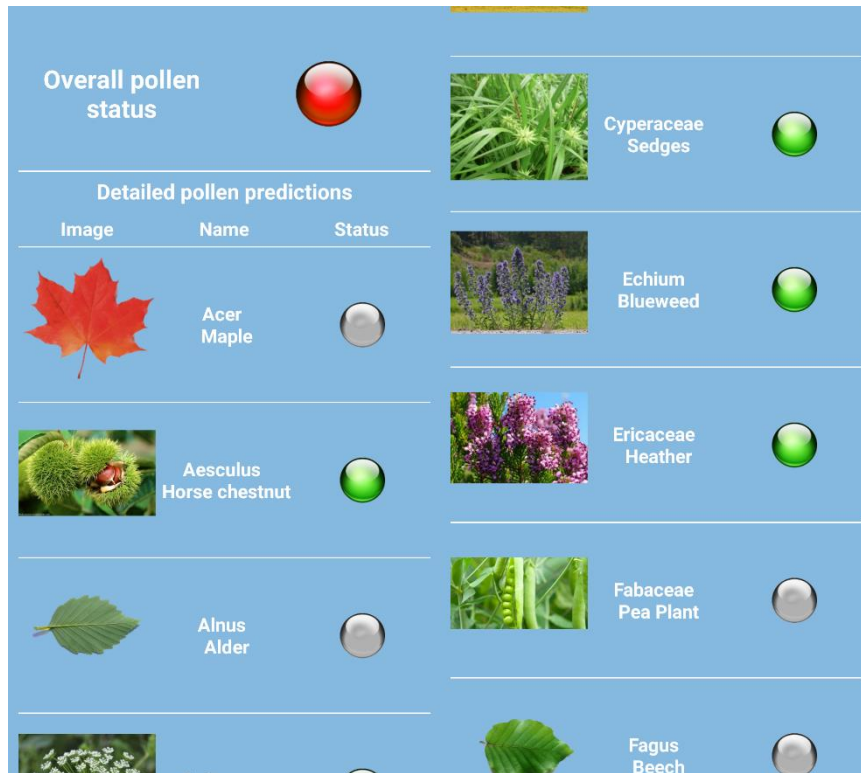


Figure 35: ScrollView screenshots

As you can see on the second screenshot there's a scrollbar present.

### 2.4.3 GraphView

In order to represent the historic data I've opted to use some charts. In order to build these charts I've used a library called GraphView. The activity where I make use of this is the graphActivity, which represents the historic data.

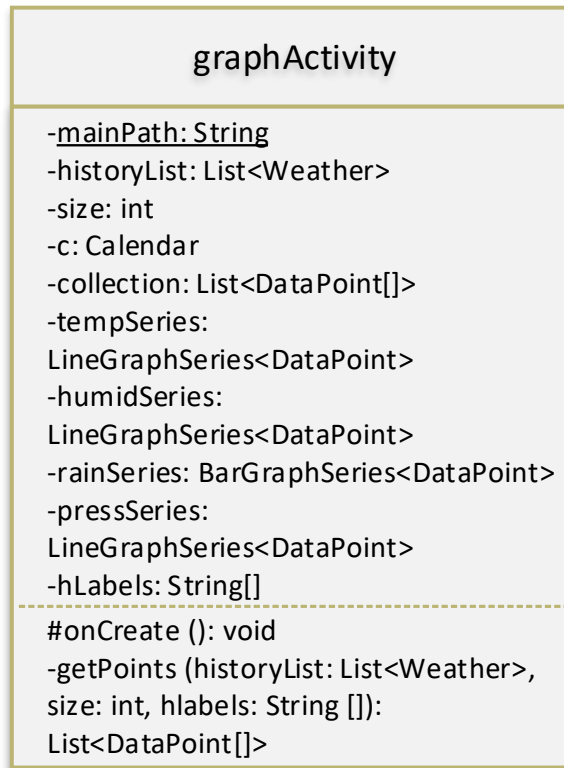


Figure 36: graphActivity UML

This class creates three charts. In order to do so it locates the XML file, containing information about the pollen, using the mainPath string which describes this location. It then passes this location to the xmlParser (using the getPollenListFromFile method) which parses the file and returns a list of Weather objects containing the information from the file. This list is saved as the historyList, the size refers to the size of this list which we'll need to transform this list into a set of DataPoint collections.

In order to fill the charts we need LineGraphSeries for line graph and BarGraphSeries for bar charts. These series are lists of DataPoint objects. A DataPoint contains 2 numbers which describe a point on the chart. The getPoints method converts this historyList into a list of collections of DataPoint objects, one collection for each chart or graph we want to draw.

We then define each collection based on which series they describe: temperature, rainfall, humidity or air pressure. The hLabels is also produced in the getPoints method, this describes the x-axis' labels which are timestamps. While the y-axis' labels are linked to the values displayed and are automatically generated.

## 3 Tests

This chapter will cover some of the problems I ran into and how I've solved them. The method of solving these problems involved some debugging by using tags or console lines which returned certain information wherever these were placed. This way I could read out what the application was doing step by step and intervene where necessary.

### 3.1 Finished downloads

The main problem I've encountered when downloading the files was finding out when the downloads are finished. When I ran the application, it didn't find any information to fill the pages with information. The cause was that it tried filling the page before the files were finished downloading which caused the page to be empty because no information was available yet.

#### *Solution*

I've solved this by implementing the broadcast receiver which looked for broadcasts sent by the downloader notifying a file has finished downloading. Only once this occurred the application could start filling the page with information.

The problem returned when I needed more than one file; I needed multiple files to obtain all the information necessary. The broadcast receiver activated for whichever file had finished downloading first. Therefore I saved the id's given to the download, one unique id for each download. Now I could identify each broadcast with a certain file. Once all of the necessary files were downloaded the application could start filling the page with information.

### 3.2 Reading the files

When trying to implement a parser for the XML files I ran into quite some troubles. Most of these troubles were in fact just typo's. I defined some tags with faulty information which caused the parser to not find the wanted tags. Another big problem was the separation between the START\_TAG's and the END\_TAG's. At first the parser only returned END\_TAG's which caused it to return faulty information or even no information. The problem was in my switch case where I forgot to add the break; statement at the end of the START\_TAG case. Therefore the parser just skipped these tags and only looked for END\_TAG's.

#### *Solution*

The solution was quite simple, once I found out. I corrected the faulty defined tags and added a break; statement at the end of each case. This resulted in the parser returning the correct information when asked for.

A second problem which occurred while trying to read the files was with the CSV type of file. The first couple of lines in these files contained for the application unnecessary information. So the first parts of information this parser returned was unusable.

#### *Solution*

In order to solve this I first implemented a for loop which does nothing the first lines where this faulty information was defined. After these lines it started reading and returning the correct information.

But, this method wasn't really clean and I went to look for a better solution. I found out that in the constructor of the CsvReader you can add an argument which defines the number of lines to skip before actually starting to read. This made my previous solution with the for loop unnecessary and still returned the correct information. I opted for this solution as it was the cleaner solution.

### 3.3 Background image

The applications background is an image which differs depending on the current or predicted weather state. I used some pictures and cut them to a ratio which is most common with Android devices. But when I used this image on a different device it was stretched messed up.

#### *Solution*

In order to solve this I used a scaler. First I measured the user's screen, then the image is rescaled to fit the screen perfectly and in a good state. Now whichever device is used the image should be properly scaled to its screen size. (24)

### 3.4 X-axis labels

When generating the charts the graphView class looks at each of the DataPoint objects and use them as coordinates. So I had to decide a way to generate the x-coordinates in order for the charts to represent correct information. Since the x-axis is about time, one DataPoint for each hour, I tried using a Date objects because the graphView knew how to handle these. This resulted in the graph itself being correct but the x-labels were unreadable. When I tried to format these Date objects the graph didn't represent the correct data because the graphView saw these formats as simple strings which messed up the order.

#### *Solution*

The solution was to set up a custom list of strings which represented the x-labels, they were yet again formatted Date objects but they have no relation to the building of the chart. This caused the charts to be correct and the labels as well.



## 4 Conclusion

This application provides an easy way to access the current weather status as well as predictions as well as some statistics. It's an easy tool to check whether you'll need a jacket when going outside or deciding when to do a barbecue in your garden.

The application has met all of the required objectives. It gathers the right sources for information and it provides the user with this information in a way where it doesn't take a lot of effort to find the information required.

This project has improved my knowledge and interests of the Android environment not only by knowing how to implement certain functionalities but mostly by solving the problems I've encountered. I've found a lot of information and help online, the Android community is a very open community. You can implement certain solutions but they don't always fit your needs. By adapting these solutions to my problems I came to understand them thoroughly. I came to recognize some things I've been learning the previous years at college but came to expand this knowledge by creating this application.

Nonetheless, this application is far from finished. Future improvements could involve expanding the different locations, this will require valuable sources of information.

All of the data acquired at this point is data straight from an online source, a possible improvement could be to save some of this data on the device in order to expand the information known about previous weather statistics as for now you can only go back 24 hours.

Another useful improvement is a widget, a widget is a miniature version of the application which can be embedded on the home screen. This way the user wouldn't have to open the application in order to receive some information nonetheless.

## 5 Appendix

### 5.1 Manual

This part covers the use of this application from installation to how the application is used.

#### Installation

First of all we have to retrieve the .apk file. This file is located on the disk provided at the end of this book. Copy the .apk file to your Android device.

This can be done by connection your device to a computer and, when prompted, select to use it to transfer files. You can now copy the .apk file through your computer to a location on the device, it's important to remember this location.

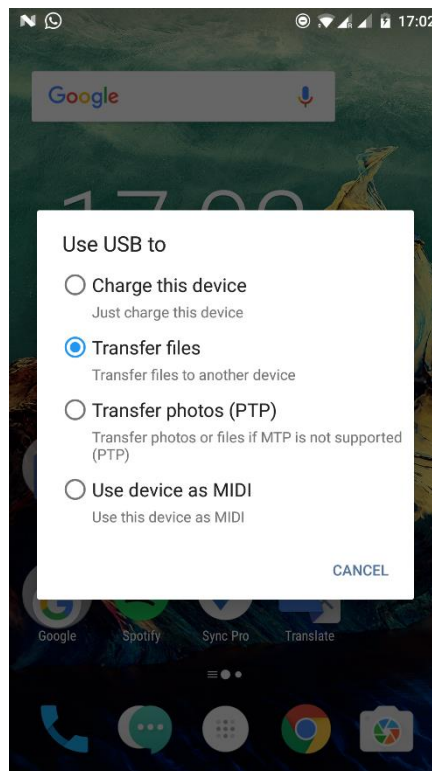


Figure 37: Device usage screenshot

Once the file is on your device, we have to enable the phone to install applications from an unknown source. In order to do so go to the Settings of your device, scroll down and choose the Security section. In here you can activate the ability to install application from an unknown source.

Once this is done, locate the .apk on your device and click it. The installation will now do its thing and when it's done, the application is installed and can be opened from your applications folder.

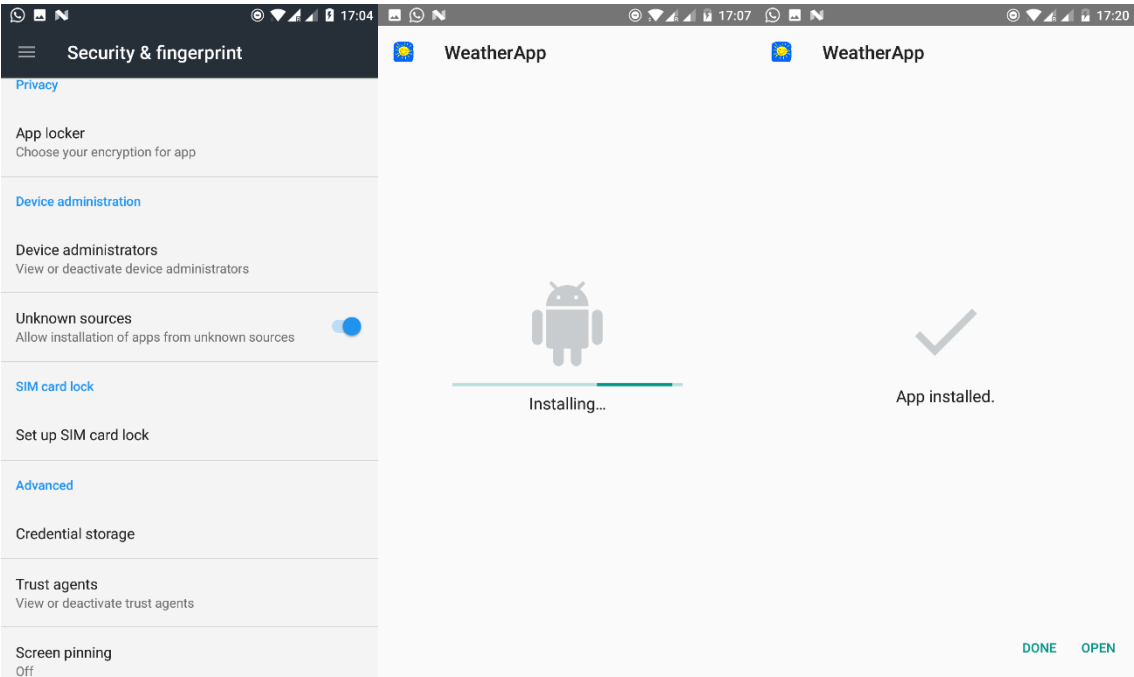


Figure 38: Installation screenshots

## How to use

Once the application is installed, we can start using it. This part will guide you through the application step by step.

You open the application by touching the applications' icon.



*Figure 39: Open application screenshot*

Once opened, the first thing you'll see is the data for the current day. The top half is current data while the bottom half shows the predictions made for today. If you swipe to the right, you'll see tomorrow's predictions if you swipe again you'll see predictions for the day after tomorrow and so on up to 7 days further.



Figure 40: Swipe screenshots

As you can see at the top of the screen, our location is currently set to Valladolid. In order to change the location we go back to the current day. At the top left we can see a location icon. Once we click this, we can select another location and the data for this location will automatically be loaded.

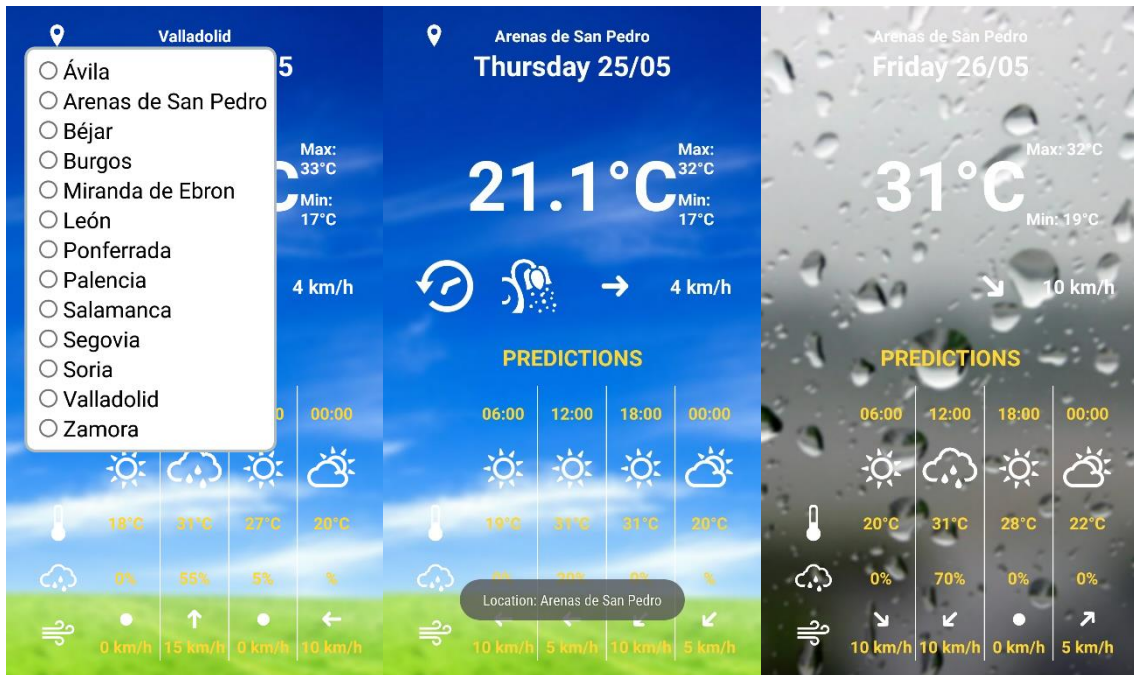


Figure 41: Select location screenshots

Once again, on today's page we have two extra icons in comparison to the other days. The one resembling a flower is for information regarding the pollen.



Figure 42: Pollen screenshot

The other one resembling a clock is for historical information 24 hours back in time. This contains 3 charts: one for the temperature, one for humidity and rain and one for the air pressure. Yet again, you can scroll through this page to see all of the charts.

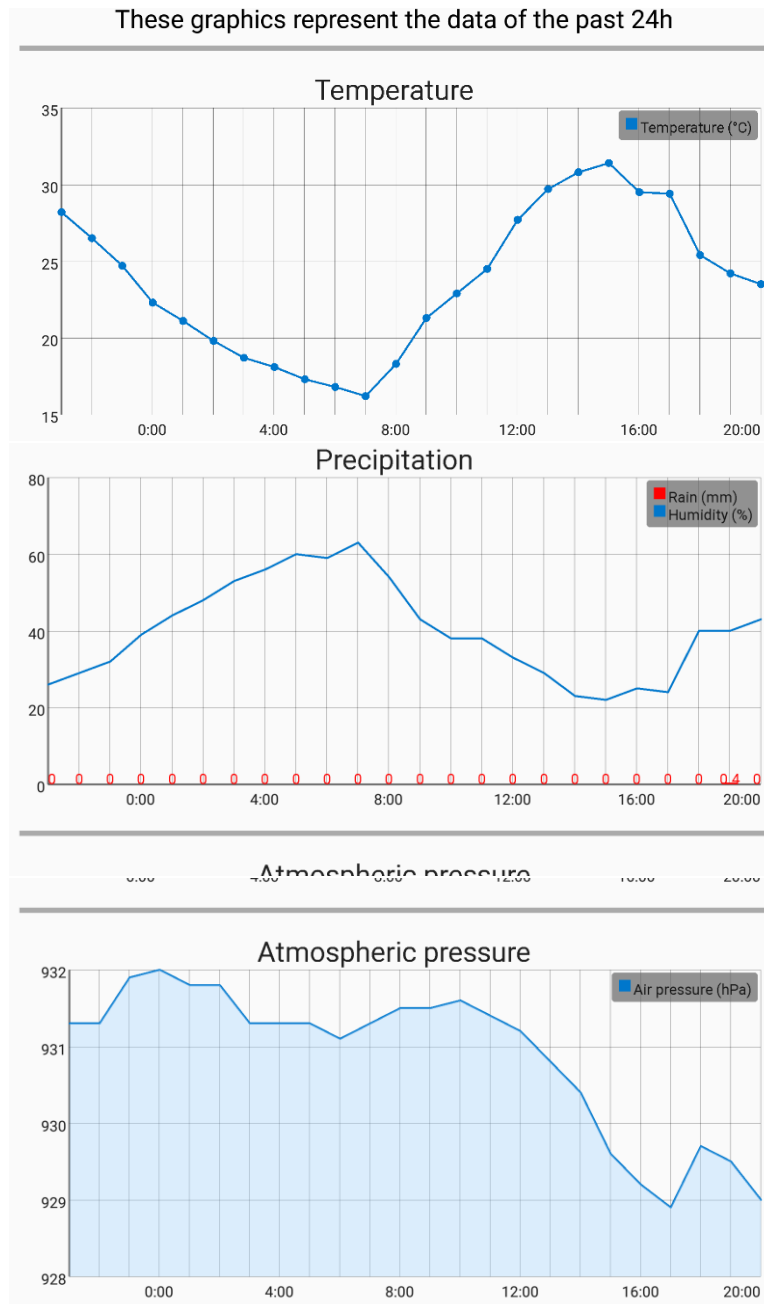


Figure 43: Charts screenshots

## 5.2 Bibliography

1. Apple Vs Android - A comparative study. *AndroidPub*. [Online] [Cited: May 25, 2017.] <https://android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683>.
2. Choosing a membership. *Apple Developer*. [Online] [Cited: May 25, 2017.] <https://developer.apple.com/support/compare-memberships/>.
3. Android (operating system). *Wikipedia*. [Online] [Cited: May 25, 2017.] [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)).
4. Junta of Castile and León. *Wikipedia*. [Online] [Cited: May 25, 2017.] [https://en.wikipedia.org/wiki/Junta\\_of\\_Castile\\_and\\_Le%C3%B3n](https://en.wikipedia.org/wiki/Junta_of_Castile_and_Le%C3%B3n).
5. ConnectivityManager getNetworkInfo(int) deprecated. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/32547006/connectivitymanager-getnetworkinfoint-deprecated>.
6. DownloadManager. *Android Developers*. [Online] [Cited: May 25, 2017.] <https://developer.android.com/reference/android/app/DownloadManager.html>.
7. Android download manager completed. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/21477493/android-download-manager-completed>.
8. Comma-separated values. *Wikipedia*. [Online] [Cited: May 25, 2017.] [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values).
9. opencsv. *opencsv*. [Online] [Cited: May 25, 2017.] <http://opencsv.sourceforge.net/>.
10. XML. *Wikipedia*. [Online] [Cited: May 25, 2017.] <https://en.wikipedia.org/wiki/XML>.
11. XML Processing in Android and Java: XMLPull Example. *VodeVoila*. [Online] [Cited: May 17, 2017.] <http://www.codevoila.com/post/64/xml-processing-in-android-and-java-xmlpull-example>.
12. How to implement a ViewPager with different Fragments/Layouts. *stackoverflow*. [Online] [Cited: May 25, 2017.] <http://stackoverflow.com/questions/18413309/how-to-implement-a-viewpager-with-different-fragments-layouts>.
13. Android ViewPager Example. *Java Code Geeks*. [Online] [Cited: May 25, 2017.] <https://examples.javacodegeeks.com/android/core/view/viewpager/android-viewpager-example/>.
14. Using ViewPager for Screen Slides. *Android Developers*. [Online] [Cited: May 25, 2017.] <https://developer.android.com/training/animation/screen-slide.html>.



15. Introduction to Activities. *Android Developers*. [Online] [Cited: May 25, 2017.] <https://developer.android.com/guide/components/activities/intro-activities.html>.
16. Android PopupWindow custom layout. *tutorialsbird*. [Online] [Cited: May 25, 2017.] <http://www.tutorialsbird.com/android-popupwindow-custom-layout/>.
17. Pop up window to display some stuff in a fragment. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/18461990/pop-up-window-to-display-some-stuff-in-a-fragment>.
18. GraphView - open source graph plotting library for Android. *GraphView*. [Online] [Cited: May 25, 2017.] <http://www.android-graphview.org/>.
19. App Manifest. *Android Developers*. [Online] [Cited: May 25, 2017.] <https://developer.android.com/guide/topics/manifest/manifest-intro.html>.
20. How to use SharedPreferences in Android to store, fetch and edit values. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/3624280/how-to-use-sharedpreferences-in-android-to-store-fetch-and-edit-values>.
21. How I get Attribute using by XMLPull parser. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/7726239/how-i-get-attribute-using-by-xmlpull-parser>.
22. Best practice for instantiating a new Android Fragment. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/9245408/best-practice-for-instantiating-a-new-android-fragment>.
23. How to use `RelativeLayout.setBackgroundDrawable()` with a bitmap? *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/7466900/how-to-use-relativelayout-setbackgrounddrawable-with-a-bitmap>.
24. Full screen background image in an activity. *stackoverflow*. [Online] [Cited: May 25, 2017.] <https://stackoverflow.com/questions/16135984/full-screen-background-image-in-an-activity>.



