**Politecnico di Milano**

Software Engineering 2

# Implementation and Test document

Version 1.0 - 30/12/2017

Installation link

Application server source code link

Android app source code link

*Pietro Melzi, Alessandro Pina, Matteo Salvadore*

AA 2017-2018

# Contents

# Chapter 1

# INTRODUCTION

## 1.1 Purpose

This document has the purpose of describing how our group has performed the implementation and test assignment, in particular which functionalities we have actually implemented and which one are missing in this first release, along with the rationale behind the choice of excluding them. It also describes the structure of the code, the adopted development frameworks, the installation instructions, and information on how we have performed our tests.
Lastly this document describes how our software is to be installed and which software tools we have used to develop our system.
This document is addressed to those who will have to validate and evaluate our implementation of Travlendar+.

## 1.2 Scope

Travlendar+ is a service based on a mobile application.
The system aims to help its users with a calendar-based application that supports them, by computing travels between their events, helping them to organize their schedule and allowing them to arrange their trips.
For further details see the Scope section in the RASD document.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- *Arrange trip*: the system provides all the information available about a travel to the user. If the travel is related to one or more public travel means, the system helps the user to organize it. It is indicated if the user already holds a valid ticket for a certain travel or if he has to buy a required one. Purchase of a ticket is handled on the websites of the transport service providers. Trip and travel are synonymous.

- *Best path*: it is the preferred travel option proposed to reach the event location among all the feasible paths, according to parameters specified by the user. The best path can be chosen according to one of these features: length, cost or environmental sustainability. Best path is the path taken into account and showed in the daily schedule; the user can substitute it at any moment with an alternative feasible path.

- *Break event*: it is an optional event whose starting and ending time are flexible. The user can define this kind of event when he wants to reserve a certain amount of time in the schedule and he doesn't need to specify a starting time. For instance, a user could be able to specify that lunch must be possible every day between 11:30- 2:30, and it must be at least half an hour long, but the specific timing is flexible. The app would then be sure to reserve, if possible, at least 30 minutes for lunch each day.

- *Constraint*: it is a rule defined on travel means by the user. When the system calculates feasible paths, each constraint must be taken into account: travel options not respectful of existing constraints are ignored. A constraint can be associated to a type of event.
  There are different types of constraints: min/max distance allowed for a travel mean, interval of hours in which it is possible to take a travel mean and possibility to deactivate a travel mean.
- *Current Unix Timestamp*: it is the number of seconds since Jan 01 1970. (UTC)
- *Customized type of event*: it is a set of rules related to a particular type of event that can be used several time by the user. The user can define a customized type of event in hs preferences, starting from the default type of event.
- *Default type of event*: it is a general set of rules defined usually the first time that the user exploits the system, it can be modified. Default type of event is related to each event that doesn't need a particular type of event. New types of event are defined starting from the constraints enclosed in the default one.
- *Feasible path*: a path that allows the user to reach a specified location before the starting time of the event to attend. It observes the constraints defined for that event.
- *Google Maps API*: a set of functions offered by Google Maps to decide the travel path between two locations.
- *Overlapping event*: an event (or a part of it) that happens in the same time slot of another event, added previously in the schedule. Because the schedule must be feasible, an event is considered as "overlapping" also if only its related travel overlaps an event present in the schedule.
- *Periodicity*: it is related to events that occur more than one time. It can be defined as weekly, monthly or in which days of the week the event happen.
- *Schedule*: a daily plan containing a set of events inserted by the user that allows him to travel and attend all the events. If an event overlaps other events, it cannot be inserted into the schedule.
- *Type of event*: a set of rules (constraints) that is defined by the user and can be associated to multiple events.
- *Transport service provider*: a public or private company that controls and supplies the transport with a travel mean.
- *Travel*: it is used to indicate the path that the user has to follows in order to reach a location. It can be composed by different travel components.
- *Travel component*: each single path that can be traveled with a travel mean. It has a starting time, a ending time, a departure location, an arrival location and a length. The union of one or more travel components creates a travel.

### 1.3.2 Acronyms

- *API*: Application Programming Interface;
- *ACID*: Atomicity, Consistency, Isolation and Durability;
- *DAO*: Data Access Object;
- *DB*: Data Base;
- *DBMS*: Data Base Management System;
- *DD*: Design Document;
- *EJB*: Enterprise Java Beans;
- *GCM*: Google Cloud Messaging;
- *GF*: GlassFish;
- *GTFS*: General Transit Feed Specification;
- *HTTP*: HyperText Transfer Protocol;
- *ICSEA*: Tenth International Conference on Software Engineering Advances;
- *IDE*: Integrated Development Environment;
- *ITD*: Implementation and Test Document;
- *Java EE*: Java Enterprise Edition;
- *JAX-RS*: Java API for RESTful Web Services;
- *JPA*: Java Persistence API;
- *RASD*: Requirement Analysis and Specification Document;
- *RESTful*: REpresentational State Transfer;
- *RSA*: Rivest-Shamir-Adleman algorithm;
- *UTC*: Coordinated Universal Time;
- *XML*: eXtensive Markup Language.

### 1.3.3 Abbreviations

- *GMaps*: Google Maps;
- *I&T*: Integration and Testing;
- *TSP*: Transport Service Provider.

## 1.4 Revision history

2017/12/30 - *Version 1.0* - I&T Delivery.

## 1.5 Reference Documents

- Specification Document: ”Mandatory Project Assignments.pdf”;
- Specification Document: ”Implementation and Testing Assignments.pdf”;
- Requirements Analysis and Specification Document, version 1.1;
- Design Document, version 1.1.

## 1.6 Document Structure

This Implementation and Test Document is composed by eight chapters:

1. The first chapter is an introduction to ITD and contains also other information: definition of the used terms, revision history of this document and the references;
2. The second chapter describes requirements and functionalities that are actually implemented in the software;
3. The third chapter includes the adopted development frameworks;
4. The fourth chapter explains how the source code is structured;
5. The fifth chapter contains information on how the testing has been performed;
6. The sixth chapter provides the installation instructions;
7. The seventh chapter contains a report of the hours spent to fulfill this assignment;
8. The eighth chapter contains references to external documents used in this document and to the software used in order to write this document and to develop our implementation of Travlendar+.

Chapter 2

# IMPLEMENTED REQUIREMENTS

Our first release of Travlendar+ includes all core functionalities of the application server and of the database server, as specified in the previous documents (more details in section 2.1 of this document), a set of initial functionalities in the Android application (more details in section 2.2) and does not include neither the web server implementation or the iOS app. In the following schema we have highlighted the components of our general architecture (see also section 2.1 of DD) that are actually implemented and tested (components are encircled in green, the integration of the subsystems are colored in green). As stated before, in the following sections we will explain in details what exactly has been implemented.

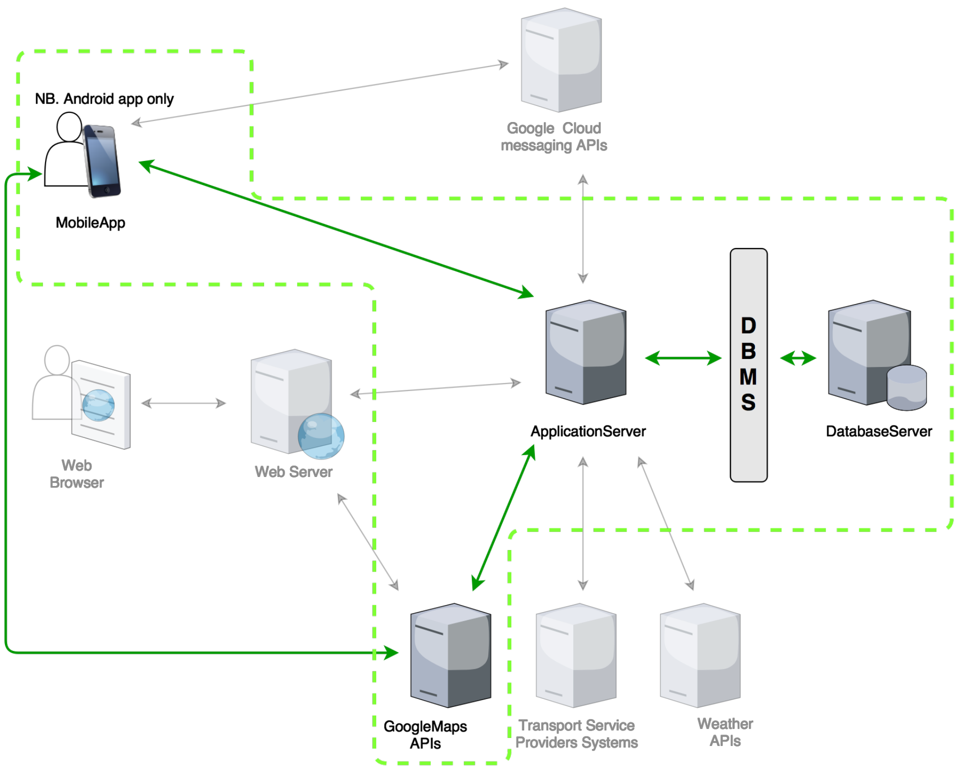


Figure 2.1: General Architecture - Implemented subsystems

Our software includes all the functionalities that:

- Allow the users to have a calendar of events which can be either scheduled or non scheduled;
- Compute feasible travels which allow the users to reach their scheduled events in the allotted time;
- Allow the users to define their own preference profiles, which are to be applied to the travels proposed by our system;
- Allow the users to save their preferred locations into their profiles;
- Allow the users to define flexible breaks and to define periodical events;
- Allow the users to save their tickets and associate them to compatible travel components.

In this first release are not included the functionalities that:

- Allow the users to buy new tickets within our application;
- Allow the users to locate the nearest sharing vehicles;
- Allow the system to take into account weather, forecast, strikes and traffic info and to notify the user if a path is no more feasible due to those issues.

We have decided not to include them in our first release since we consider them as nice-to-have feature, but not mandatory for an initial release. They depend on other functionalities we have implemented and so they must come after those functionalities. We had then to choose how to allocate properly the limited amount of time given for the implementation phase and those functionalities would have required more time to be implemented.
Here we provide a list of use cases, taken from our RASD document (section 3.2.3) that are satisfied in this first release:

UC1 Registration;

UC2 Login;

UC3 View calendar;

UC4 Create event;

UC5 Create personalized event profiles;

UC6 Define flexible breaks;

UC7 Arrange trips - Only with ticket inserted by the user;

UC9 Add ticket possessed;

UC10 Obtain feasible travel paths - Without the alternatives paths;

UC11 Choose between overlapping events;

UC12 Edit event - Only in Application server, not yet available on Android app;

UC13 Delete event;

UC14 Edit personalized event profiles;

UC15 Delete personalized event profiles.

## 2.1 Application Server and Database Server

In this section we specify which requirements are actually implemented in our ApplicationServer subsystem, examining every sub-component, and stating which requirements are implemented in this first release of Travlendar+. We also state the motivations for including them and excluding others (see also chapter 5 of DD - requirement traceability).

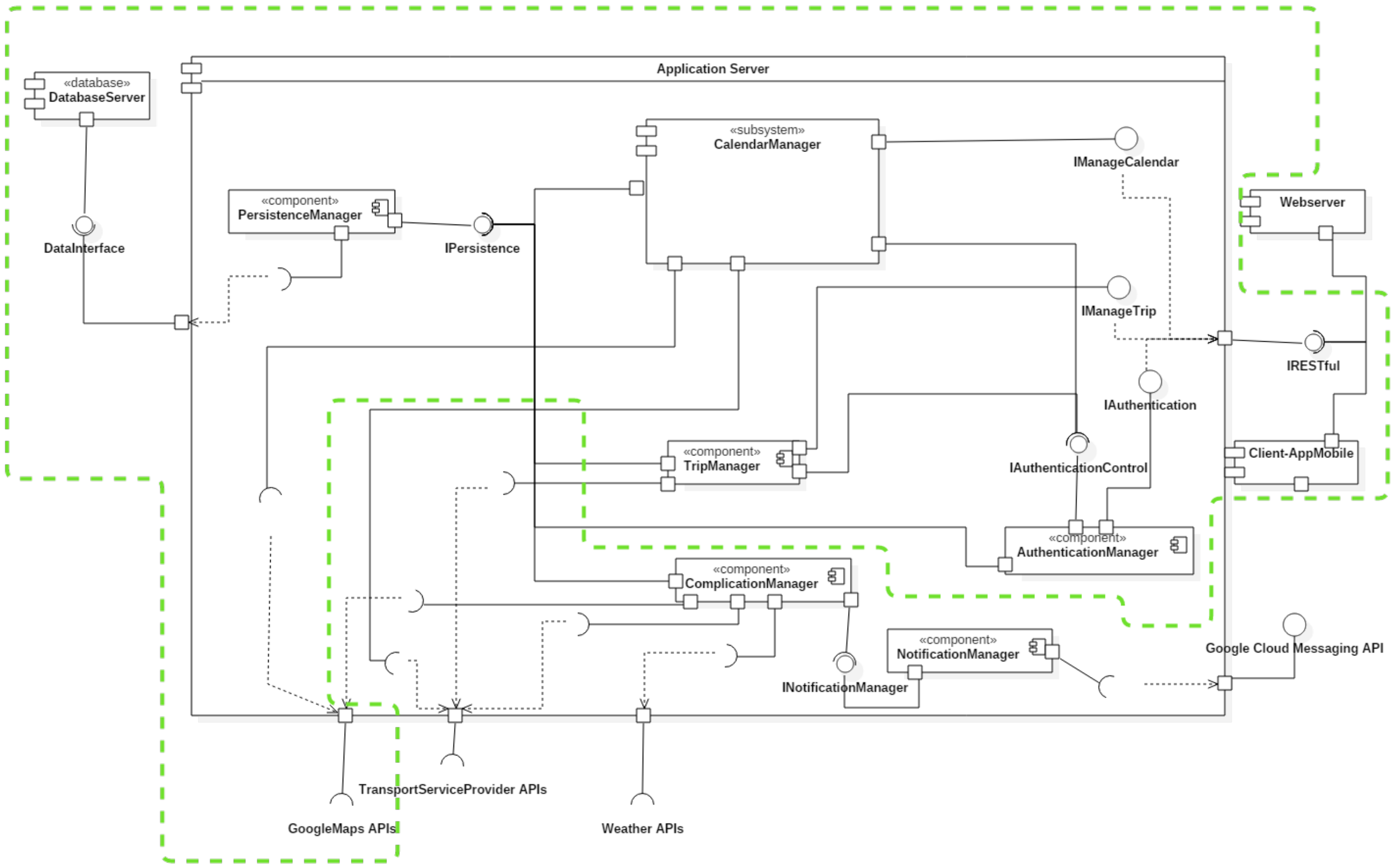


Figure 2.2: The implemented components of the Application Server

### 2.1.1 Authentication Manager

All the requirements related to the authentication manager are fulfilled in this first release: the user can register himself into the system, log in from any device he prefers (but only one at the time), modify his account's info, delete his account and request a new password.
As specified in the previous documents, these are the requirements met by this manager:

- **[R1]** the system checks if the e-mail inserted is real;
- **[R2]** a user cannot sign up with the same e-mail twice;
- **[R3]** the e-mail and password inserted must be correct;
- **[R4]** incorrect credentials prevent the user from logging in;
- **[R38]** a user must be logged.

### 2.1.2 Event Manager

Event manager handles insertion, deletion and modification of both events and break events into the user's calendar. It interacts with both path and schedule manager in order to provide feasible paths to the events and to decide whether they can be scheduled or not. In the application server also the propagation of periodical events through the time is handled properly.
In our system is not yet handled the possibility to modify and delete with one operation all the periodic events generated from one single event, due to time-related issues. In our system is not yet handled the possibility to define events without an ending time since we do not consider this feature essential, but just an optional and not so necessary feature.
As specified in the previous documents, these are the requirements met by this manager:

- [**R5**] a user must specify all mandatory fields to add the new event;
- [**R6**] the system reserves the specified time-slot for the event;
- [**R7**] the system warns the user if the inserted event overlaps with an already existing one;
- [**R29**] the system allows the user to specify a flexible interval and a minimum amount of time to schedule a break.

And these are the requirements not yet implemented in this manager:

- [**R8**] if the ending time of an event is not specified, the systems considers as ending time the hour of departure for the successive event;
- [**R9**] when an event is inserted after an event without a specified ending time, the ending time of the first event is anticipated as stated in [R8].

### 2.1.3 Preference Manager

All the requirements related to the preference manager are fulfilled. This module offers all the functionalities needed to insert, modify and delete the user's event profiles and the user's preferred locations. It is also able to apply the user's preferences to the feasible travels computed by our system.
As specified in the previous documents, these are the requirements met by this manager:

- [**R12**] the system does not consider paths that violate constraints on travel means defined by the user;
- [**R13**] the system checks user preferences to decide which feasible travel path is the best;
- [**R15**] appropriate travel means must be suggested according to the type of event that they are related to;
- [**R23**] the system requires minimum and maximum length allowed for a path to impose a constraint on a travel mean;
- [**R24**] the system requires an interval of time allowed to impose a constraint on a travel mean;
- [**R25**] the system does not consider solutions that violate constraints;
- [**R26**] the system allows the user to specify one or more travel means that cannot be used;
- [**R27**] the system does not consider solutions that include deactivated travel means.

### 2.1.4 Path Manager

This module manages the feasible travels computation, after the insertion or the modification of an event. It interacts with the *PreferenceManager* in order to obtain the information needed to guarantee that the user preferences are respected in every proposed path.
The functionalities that allow to change a selected path, choosing between a set of feasible alternatives, to obtain info that allow to draw a feasible path on a map and to obtain also options that include sharing vehicles are not yet available; we've chosen not to include them due to time-related issues and because they would have been nice-to-have features but not essential for an initial release.
As specified in the previous documents, these are the requirements met by this manager:

- [**R10**] every travel path proposed must be feasible in the available time (the interval between two consecutive events);
- [**R11**] if the travel involves two or more travel means, the starting location of the first proposed travel path and the ending location of the last proposed travel path must coincide respectively with the starting location and the ending location of the whole planned travel;
- [**R12**] the system does not consider paths that violate constraints on travel means defined by the user;
- [**R13**] the system checks user preferences to decide which feasible travel path is the best;
- [**R15**] appropriate travel means must be suggested according to the type of event that they are related to;
- [**R25**] the system does not consider solutions that violate constraints;
- [**R27**] the system does not consider solutions that include deactivated travel means;
- [**R35**] the system provides information about time of departure and arrival of the proposed travels.

And these are the requirements not yet implemented in this manager:

- [**R18**] the system must show to the user all possibilities to reach a location in according with the requirements of [G4];
- [**R19**] alternative feasible travel paths must not generate overlappings with other events of the schedule;
- [**R28**] for each travel path, the system estimates its carbon footprint produced;
- [**R37**] the system provides information about travel time with shared vehicles.

### 2.1.5 Schedule Manager

This module is able to check if an event overlaps with other events and to guarantee that the flexible breaks are respected. It also checks that the user's travels do not overlap with other events. When an event overlaps with another one, the schedule manager puts it in a separate list of non scheduled events and it manages the user's requests of rescheduling those events.
As specified in the previous sections, events without an ending time and alternative travels are not yet implemented into our system since, due to time-related issues, we've decided to exclude them from our first release of Travlendar+.
As specified in the previous documents, these are the requirements met by this manager:

- [**R6**] the system reserves the specified time-slot for the event;
- [**R7**] the system warns the user if the inserted event overlaps with an already existing one;
- [**R14**] the system warns the user if it is not possible to arrive at an event location before its starting time;
- [**R20**] the combination of the travel paths proposed for the day must be feasible in the allotted time;
- [**R21**] if there are multiple events at the same time the system will propose in the schedule only the first event added;
- [**R22**] if the user forces into the schedule an event that overlaps with events already present in the schedule, these are removed from the schedule;
- [**R30**] if there is enough time for a break, the system reserves it within the specified flexible interval;
- [**R31**] if there is not enough time into the flexible interval specified, a warning is thrown.

And these are the requirements not yet implemented in this manager:

- [**R8**] if the ending time of an event is not specified, the systems considers as ending time the hour of departure for the successive event;
- [**R9**] when an event is inserted after an event without a specified ending time, the ending time of the first event is anticipated as stated in [R8];
- [**R19**] alternative feasible travel paths must not generate overlappings with other events of the schedule.

### 2.1.6 Trip Manager

Trip manager is the last module we have implemented and offers only the functionalities that allow the user to save and visualize his tickets and to associate them to travel components for which they are applicable.
This module does not yet provide functionalities that allow the user to buy public transportation tickets and to locate the nearest sharing vehicles, since they would have required the integration of other external APIs and we have decided to allocate our time in other functionalities that we consider more important than these ones.
As specified in the previous documents, these are the requirements met by this manager:

- [**R32**] the system allows the user to specify all the ticket he already owns;
- [**R33**] the system shows to the user if he already holds a ticket for a proposed travel;

And these are the requirements not yet implemented in this manager:

- [**R34**] the system allows the user to buy public transportation tickets according to proposed travels;
- [**R36**] the system shows to the user where sharing vehicles are located.

### 2.1.7 Complication manager and Notification Manager

Both Complication and Notification manager are not implemented in this first release of Travlendar+, since we have considered their functionalities not essential for the first release of our system and since we had to make a choice due to limited implementation time.

## 2.2 Android App

In this section we specify which requirements are actually implemented in our first release of Travlendar+'s Android app. Our Android application implements all the functionalities that are offered by our application server (for more details please refer to the previous sections) except:

- periodical events insertion;
- modifications of already inserted events and break events, only their deletion is available;
- modifications of already inserted tickets, only their deletion is available;
- user profile deletion and modification.

Furthermore the map interface displays only the user's preferred locations.
These are the requirements met by our Android application:

- [**R1**] the system checks if the e-mail inserted is real;
- [**R2**] a user cannot sign up with the same e-mail twice;
- [**R3**] the e-mail and password inserted must be correct;
- [**R4**] incorrect credentials prevent the user from logging in;
- [**R5**] a user must specify all mandatory fields to add the new event;
- [**R6**] the system reserves the specified time-slot for the event;
- [**R7**] the system warns the user if the inserted event overlaps with an already existing one;
- [**R10**] every travel path proposed must be feasible in the available time (the interval between two consecutive events);
- [**R11**] if the travel involves two or more travel means, the starting location of the first proposed travel path and the ending location of the last proposed travel path must coincide respectively with the starting location and the ending location of the whole planned travel;
- [**R12**] the system does not consider paths that violate constraints on travel means defined by the user;
- [**R13**] the system checks user preferences to decide which feasible travel path is the best;
- [**R14**] the system warns the user if it is not possible to arrive at an event location before its starting time;
- [**R15**] appropriate travel means must be suggested according to the type of event that they are related to;
- [**R20**] the combination of the travel paths proposed for the day must be feasible in the allotted time;

- **[R21]** if there are multiple events at the same time the system will propose in the schedule only the first event added;
- **[R22]** if the user forces into the schedule an event that overlaps with events already present in the schedule, these are removed from the schedule;
- **[R23]** the system requires minimum and maximum length allowed for a path to impose a constraint on a travel mean;
- **[R24]** the system requires an interval of time allowed to impose a constraint on a travel mean;
- **[R25]** the system does not consider solutions that violate constraints;
- **[R26]** the system allows the user to specify one or more travel means that cannot be used;
- **[R27]** the system does not consider solutions that include deactivated travel means;
- **[R29]** the system allows the user to specify a flexible interval and a minimum amount of time to schedule a break;
- **[R30]** if there is enough time for a break, the system reserves it within the specified flexible interval;
- **[R31]** if there is not enough time into the flexible interval specified, a warning is thrown;
- **[R32]** the system allows the user to specify all the ticket he already owns;
- **[R33]** the system shows to the user if he already holds a ticket for a proposed travel;
- **[R35]** the system provides information about time of departure and arrival of the proposed travels;
- **[R38]** a user must be logged.

And these are the requirements not yet implemented in our Android application:

- **[R8]** if the ending time of an event is not specified, the systems considers as ending time the hour of departure for the successive event;
- **[R9]** when an event is inserted after an event without a specified ending time, the ending time of the first event is anticipated as stated in [R8];
- **[R16]** if a strike occurs, the system will not consider travel means involved in it;
- **[R17]** if the weather forecasts rain or other adverse conditions, the system will not consider paths involving the bicycle;
- **[R18]** the system must show to the user all possibilities to reach a location in according with the requirements of [G4];
- **[R19]** alternative feasible travel paths must not generate overlappings with other events of the schedule;
- **[R28]** for each travel path, the system estimates its carbon footprint produced;
- **[R34]** the system allows the user to buy public transportation tickets according to proposed travels;
- **[R36]** the system shows to the user where sharing vehicles are located;
- **[R37]** the system provides information about travel time with shared vehicles;

Chapter 3

# ADOPTED DEVELOPMENT FRAMEWORKS

In the implementation phase we adopted the choices stated in *section 2.3.1 (Recommended Implementation Choices)* of the *Design Document* for the components that we have actually developed: Application Server and Android Application along with their relative databases.

## 3.1 Adopted programming languages

### 3.1.1 Java

We chose Java as programming language in order to build the Application Server with Java EE, a set of specifications used to develop enterprise applications. Java EE is suitable for our needs and it is part of the arguments taught during laboratory lessons.
For what Java concerns, we can state that the advantages are greater than the disadvantages. The main qualities of Java are:

- *object-oriented*: the programming code is based on the concept of object. Each object can contain several fields (attributes) and functionalities (methods). The code is widely reusable and it is possible to establish a sort of hierarchy and relations between the different objects. Travlendar+, moreover, is made up of different entities, linked to each other, that can be easily represented as objects;
- *simple*: syntax is easy to understand and does not include confusing features, such as explicit pointers or operator overloading. Also, unreferenced objects are automatically removed by Java Garbage Collector;
- *platform-independent*: the ability to be used in different platforms. Java code is converted in bytecode and follows the Write Once Run Anywhere principle. In this case the code runs on a Linux-based server, but it is written and tested at first in Windows environments.

As a drawback, Java is significantly slower and more memory-consuming than natively compiled languages such as C or C++.

About Java EE, we can confirm what we wrote in Design Document:
*"The Application Server may be implemented using Java Enterprise Edition (JEE) 8; this choice could allow the developers to focus on the logic to be provided while being supported by reliable APIs and tools, in order to reduce the complexity of the development phase. Using JEE 8 the developers could also guarantee the main non-functional requirements, which are stated in RASD document".*

In fact, we observe that Java EE includes a set of services and APIs that can facilitate quick development of stable and powerful enterprise level applications. Java EE is widespread among developers and it is also supported by different test tools and frameworks.
However, Java EE requires a heavier server than other possible solutions and for some issues that we encountered during the implementation, the documentation available online is quite poor.

### 3.1.2 Java for Android

Java for Android is, of course, the same programming language described in the previous section, but Android is a different platform and uses a different virtual machine (Dalvik).
Java is the programming language for the Android SDK.

To develop an application for Android devices the two main language possibilities are Java and Kotlin, the latter has just recently been added by Google as an officially supported language. The two languages are compatible and can be used side by side, but in this case we decided that Java was enough to realize the implementation of this project.

Kotlin offers some advantages like more succinct code, but given the limited amount of time for the implementation, we decided, in order to avoid confusion by inserting a previously not known language, to stick with Java, a programming language already known by all the members of the group.

One of the main challenges of coding in Android is the support needed by multiple versions of the operating system: it is difficult to find a compromise between reaching a big amount of devices and effectively supporting different platform versions.

### 3.1.3 XML

XML is a mark-up language used in the Application Server to describe properties or dependencies of the project. The main advantages about XML we found in the implementation phase are:

- it supports Unicode, allowing almost any information in any written human language to be communicated;
- it can represent common computer science data structures: records, lists and trees;
- its self-documenting format describes structure and field names as well as specific values;
- the strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient and consistent;
- the hierarchical structure is suitable for most types of documents;
- it is platform-independent, thus relatively immune to changes in technology.

Instead, the found XML disadvantages are:

- XML syntax is redundant or large relative to binary representations of similar data, especially with tabular data;
- hierarchical model for representation is limited in comparison to an object-oriented graph;
- encourages non-relational data structures (non-normalized data).

### 3.1.4 XML for Android

XML is used in an Android application in activity layouts, defining the visual structure for a user interface.

Layouts can be declared in two ways:

- By using XML;
- By instantiating them at runtime.

The advantage to declaring them in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior.
UI descriptions are external to the application code, meaning that they can be modified without altering your source code and recompile them. This is helpful when it comes to debug problems.

The XML vocabulary for declaring UI elements follows structure and naming of classes and methods.
In this way, you can quickly design UI layouts and the screen elements they contain, with a series of nested elements. Each layout file must contain exactly one root element.

## 3.2 Adopted middlewares

### 3.2.1 MySQL

MySQL is a very popular and open source DBMS, it is easy to install and configure and we used it to manage the server database. Moreover MySQL doesn't present particular disadvantages. We have linked MySQL with our Application Server by means of JPA, and we did not encounter relevant difficulties while performing this integration.

### 3.2.2 SQLite and Room

SQLite is a relational DBMS widely used as embedded database software for local/client storage in application software. It is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax.

In this implementation we also used the Room persistence library, which provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

There are three major components in Room:

- *Database*: Contains the database holder and serves as the main access point for the underlying connection to the relational data;
- *Entity*: Represents a table within the database;
- *Data Access Object (DAO)*: Contains the methods used to access the database.

### 3.2.3 GlassFish

GlassFish is the server used to run our server-side application, it is open source and allows the execution of Java EE functions. By means of EJB APIs, it is possible to perform security functions. By means of JAX-RS APIs, moreover, it is possible to provide RESTful services. Unfortunately, we encountered some difficulties related to GlassFish:

- we have installed GlassFish 5.0 on a virtual server with a low amount of RAM: 1GB. GlassFish 5.0 require 1GB of RAM (but 2GB are recommended) to run, so it often goes down due to our virtual server hardware specifications. We solved this problem increasing the virtual server RAM with a swap in secondary memory;
- GF 5.0 is relatively recent and contains some bugs that will be fixed in the following updates. For instance, we found that GF 5.0 is incompatible with JodaTime library.

## 3.3 Test frameworks

We have used:

- *Junit* in order to test single classes of our project that do not contain DB dependencies or injected elements. With Junit we can test single functionalities, both in general and borderline cases, and find errors in quite a short time;
- *Arquillan* in order to test the integration of the JPA entity classes with a test database, to check that their interaction with the database works properly; these tests allow us to check that every modification we perform on the entity classes work properly;
- *Mockito* in order to "mock" (simulate) functionalities of components that are yet tested or that will be tested in a different moment. The possibilities offered by function-mocking are very useful if you want to test particular instructions in the code or specific execution cases. In the project we used Mockito in order to simulate the functions offered by different EJBs. It is not commonly used for the purpose we pursued, for this reason it has been quite challenging to find a way in which to configure properly the test environment.

## 3.4 Additional APIs (not included into DD)

These are the utilized APIs not yet presented:

- *Json*: it is the format of Google Maps API's responses. JSON functions are used to retrieve the needed information from Google Maps API's response;
- *Retrofit*: it is a type-safe REST client for Android and Java. It ships with Gson by default, mapping automatically the response objects received from the server in previously defined classes;
- *Firebase Cloud Messaging (FCM)*: it is a cross-platform solution for messages and notifications. It was used in this Android app to obtain an identification code for the device in use;
- *Stetho*: it is a debug bridge for Android applications. It was used in the Android app to access the local DB;
- *Google Play Services Places*: it is an API that provides detailed information about 100 million places across the world. It was used to allow the user to choose locations and to obtain their coordinates.
- *Gson*: it is used to create a string in JSON format from the instance of an object. The string is used to fill the fields in the responses sent back the client;

# Chapter 4

# SOURCE CODE STRUCTURE

In this chapter we explain how our source code is structured.
In our GitHub repository we have put both Application Server and Android app source code into two different folders contained into the Implementation folder: link.
Since they are structured as separate projects we will explain their internal structure in two different sections:

## 4.1 Android app

```
/
└─ app
   ├─ src
   │  ├─ main
   │  │  ├─ java/it/polimi/travlendarplus ............ main Android app's package
   │  │  │  ├─ activity .........................contains activities related to layouts
   │  │  │  │  ├─ fragment ............................... contains fragments classes
   │  │  │  │  ├─ handler ................contains classes that handle server responses
   │  │  │  │  │  ├─ event ........................ handlers related to event responses
   │  │  │  │  │  ├─ location .................. handlers related to location responses
   │  │  │  │  │  ├─ preference .............. handlers related to preference responses
   │  │  │  │  │  └─ ticket ......................handlers related to ticket responses
   │  │  │  │  ├─ listener .........contains listeners that allow click and drag actions
   │  │  │  │  └─ tasks ....................contains classes that write on the local DB
   │  │  │  ├─ database ...........contains components of the local DB used by Room
   │  │  │  │  ├─ converters .................converters from boxed to primitive data
   │  │  │  │  ├─ dao
   │  │  │  │  ├─ entity
   │  │  │  │  └─ viewModel ................contains classes that store UI-related data
   │  │  │  └─ retrofit .........contains classes used to perform requests to the server
   │  │  ├─ res .............................................contains all the resources
   │  │  └─ AndroidManifest.xml ................provides information about your app
   │  └─ test/java/it/polimi/travlendarplus
   └─ build.gradle ....................................defines build configurations
```

## 4.2 ApplicationServer

```
/
├── src
│   ├── main
│   │   ├── java/it/polimi/travlendarplus ........... main ApplicationServer's package
│   │   │   ├── RESTful ..................................... contains all REST interfaces
│   │   │   │   ├── RESTfulCalendar ................. contains all calendar REST interfaces
│   │   │   │   ├── authenticationManager ..... contains all authentication REST interfaces
│   │   │   │   └── messages .................. contains all exchanged messages with clients
│   │   │   │       ├── authenticationMessages
│   │   │   │       ├── calendarMessages
│   │   │   │       └── tripMessages
│   │   │   ├── beans ............................contains all the java beans source code
│   │   │   │   ├── calendarManager .............source code of calendar related Java beans
│   │   │   │   │   ├── googleMapsUtilities ................ source code of GMaps requests
│   │   │   │   │   └── support ............ source code of path computation support classes
│   │   │   │   ├── emailManager ................. source code that handle email forwarding
│   │   │   │   └── tripManager .....................source code of trip related Java beans
│   │   │   ├── entities ...............................source code of JPA entity classes
│   │   │   │   ├── calendar
│   │   │   │   ├── preferences
│   │   │   │   ├── tickets
│   │   │   │   ├── travelMeans
│   │   │   │   └── travels
│   │   │   └── exceptions .....................................custom exception classes
│   │   │       ├── authenticationExceptions
│   │   │       ├── calendarExceptions
│   │   │       ├── encryptionExceptions
│   │   │       ├── googleMapsExceptions
│   │   │       ├── persistenceExceptions
│   │   │       └── tripManagerExceptions
│   │   └── resources
│   │       └── META-INF
│   │           ├── beans.xml ........................... configuration file of bean classes
│   │           └── persistence.xml ............................configuration file of JPA
│   └── test
│       ├── java/it/polimi/travlendarplus ......................... main test package
│       │   ├── beans ...............................contains all the Java beans test code
│       │   │   └── calendarManager
│       │   │       ├── googleMapsUtilities
│       │   │       ├── pathManager
│       │   │       ├── preferenceManager
│       │   │       ├── scheduleManager
│       │   │       └── support
│       │   └── entities ..........................contains all the JPA entities test code
│       │       ├── calendar
│       │       ├── preferences
│       │       ├── tickets
│       │       ├── travelMeans
│       │       └── travels
│       ├── resources-glassfish-embedded ..contains glassfish-embedded arquillian profile
│       │   config files
│       └── resources ...................................contains arquillian config files
└── pom.xml ............................ .. contains necessary info to build the project
```

# Chapter 5

# TESTING

In this section we describe how we have performed the testing of our system. We have basically followed the guidelines already defined in our Design document so, if you want additional information, please refer to sections 6.4 and 6.5 of that document.
As regards ApplicationServer subsystem, since in the DD we have described the testing strategy of the entire planned subsystem, we do not have performed the testing of the entire system but only until event EventManager integration (please refer to our design document to see the order of integration).

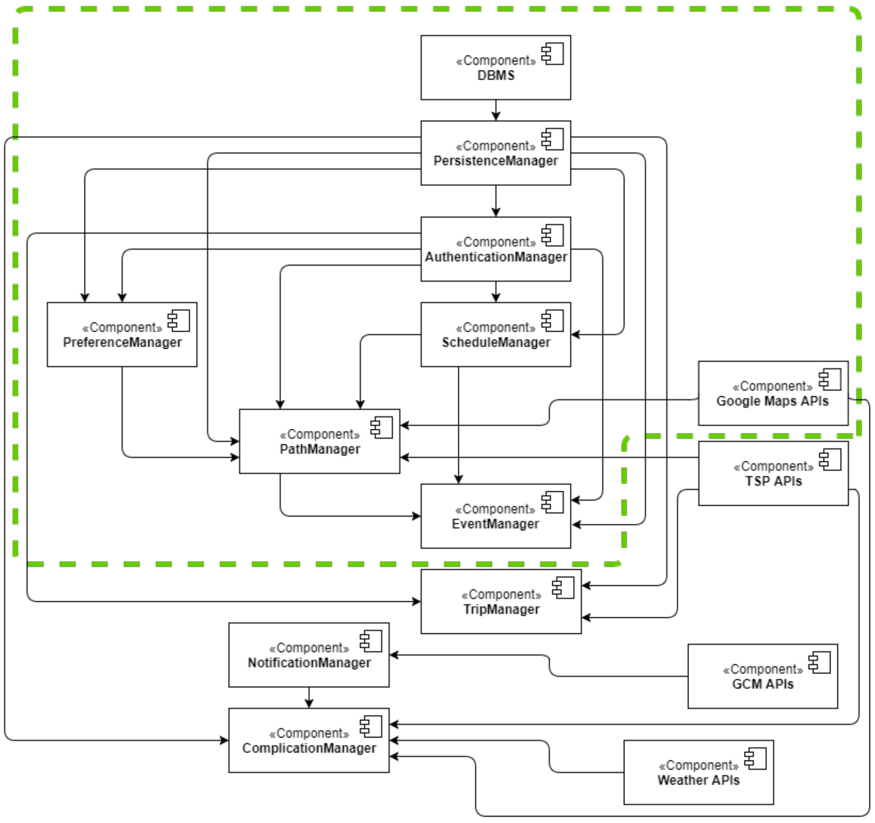

«Component» DBMS
«Component» PersistenceManager
«Component» AuthenticationManager
«Component» PreferenceManager
«Component» ScheduleManager
«Component» Google Maps APIs
«Component» PathManager
«Component» TSP APIs
«Component» EventManager
«Component» TripManager
«Component» NotificationManager
«Component» GCM APIs
«Component» ComplicationManager
«Component» Weather APIs

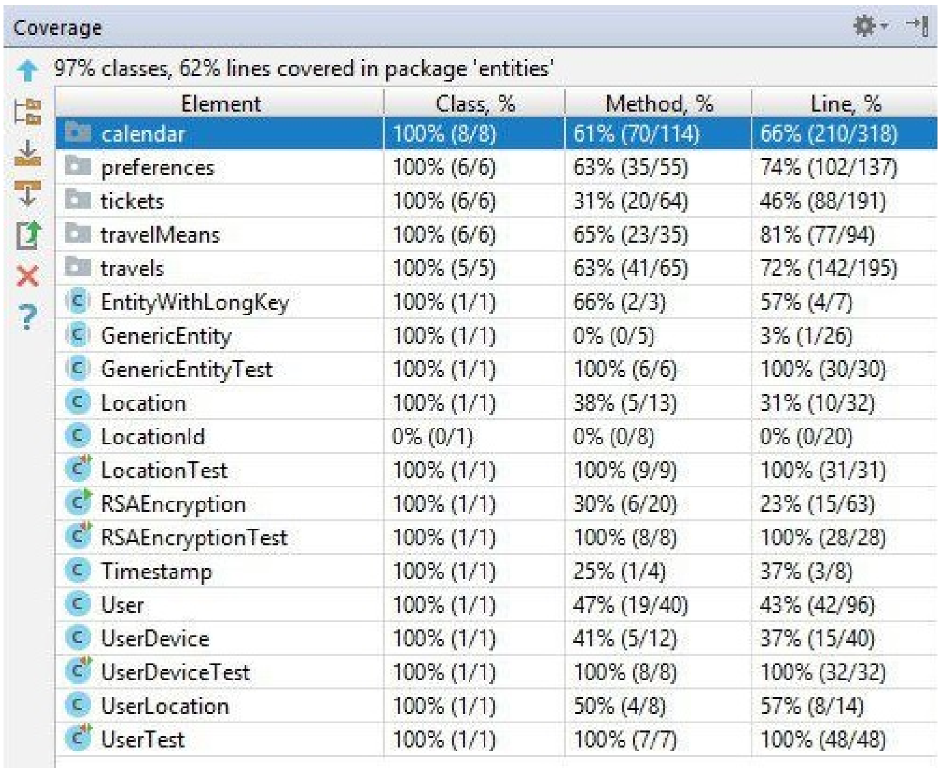
Figure 5.1: ApplicationServer - Actually tested subsystems subsystems

For what concerns the Android App subsystem we have tested the DAOs that interact with the local DB. These tests can be found in test/java/it/polimi/travlendarplus, as can be seen in the previous chapter (see section 4.1).

## 5.1 Procedure Adopted

### 5.1.1 Persistence Manager (JPA - DBMS interaction): Arquillian

In order to perform our *PersistenceManager*'s testing we have adopted the following procedure: we have exploited Arquillian features to obtain a testing configuration that allow us to use a testing container of GlassFish that can run and connect to an embedded GlassFish 3.1 instance. Our *PersistenceManager*'s tests simply persist a set of sample entities to the database, connected with the GlassFish container, and then retrieve them using JPA Criteria API. Each test method checks that the sample data retrieved from the database is correct. Before and after a test is performed, the database is cleaned up, in order to guarantee that each test is run in a clean environment. This is the outcome of these tests:


Coverage

97% classes, 62% lines covered in package 'entities'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| calendar | 100% (8/8) | 61% (70/114) | 66% (210/318) |
| preferences | 100% (6/6) | 63% (35/55) | 74% (102/137) |
| tickets | 100% (6/6) | 31% (20/64) | 46% (88/191) |
| travelMeans | 100% (6/6) | 65% (23/35) | 81% (77/94) |
| travels | 100% (5/5) | 63% (41/65) | 72% (142/195) |
| EntityWithLongKey | 100% (1/1) | 66% (2/3) | 57% (4/7) |
| GenericEntity | 100% (1/1) | 0% (0/5) | 3% (1/26) |
| GenericEntityTest | 100% (1/1) | 100% (6/6) | 100% (30/30) |
| Location | 100% (1/1) | 38% (5/13) | 31% (10/32) |
| LocationId | 0% (0/1) | 0% (0/8) | 0% (0/20) |
| LocationTest | 100% (1/1) | 100% (9/9) | 100% (31/31) |
| RSAEncryption | 100% (1/1) | 30% (6/20) | 23% (15/63) |
| RSAEncryptionTest | 100% (1/1) | 100% (8/8) | 100% (28/28) |
| Timestamp | 100% (1/1) | 25% (1/4) | 37% (3/8) |
| User | 100% (1/1) | 47% (19/40) | 43% (42/96) |
| UserDevice | 100% (1/1) | 41% (5/12) | 37% (15/40) |
| UserDeviceTest | 100% (1/1) | 100% (8/8) | 100% (32/32) |
| UserLocation | 100% (1/1) | 50% (4/8) | 57% (8/14) |
| UserTest | 100% (1/1) | 100% (7/7) | 100% (48/48) |

Figure 5.2: *PersistenceManager* - code coverage

### 5.1.2 Enterprise Java Beans: Mockito

We used Mockito to test the *PathManager*, a Stateless EJB that requires other Stateless EJBs (*ScheduleManager* and *PreferenceManager*) in order to perform its tasks. Using Mockito, all the functions performed by one of the injected beans were mocked: the idea is to consider these functions as working and to test the instructions properly defined in the *PathManager*.

In particular, we mocked *ScheduleManager* functions in this way:

- for each function that returns an event a predefined event is returned, depending on the parameter passed to the function;
- for *PathCalculation* function a random response is returned, based on the travels passed to the function;
- for save functions a *doNothing* method is used.

*PreferenceManager* functions are mocked in this way:

- *checkConstraints* function return always *true*;
- *findBestPath* function always return the first element in the array of possibilities.

The most challenging functionalities that our system offers are the calculation of feasible paths, related to the events, that respect the user's constraints and the possibility to force not scheduled events into the user's schedule.

*CalculatePath* function, for a given event, returns the path that allows the user to reach his event and the path that allows him to reach also the following one, according to the commitments in the schedule.
We considered a general case in which both previous and following events exist and we verified that:

- when an event is inserted and the allotted time between the previous event and the inserted one is enough, we expect to obtain a feasible path;
- when an event is inserted and the allotted time between the inserted event and the following one is enough, we expect to obtain a feasible path;
- when an event's preference asks that the feasible path starts from the precedent user's location (location of the previous event), we expect to obtain a path that starts form there;
- when an event's preference asks that the feasible path starts from a given location (specified by the user) we expect to obtain a path that starts form there;
- for every obtained path, related to a given event, we always expect that the path's ending location is the same as the event's location.

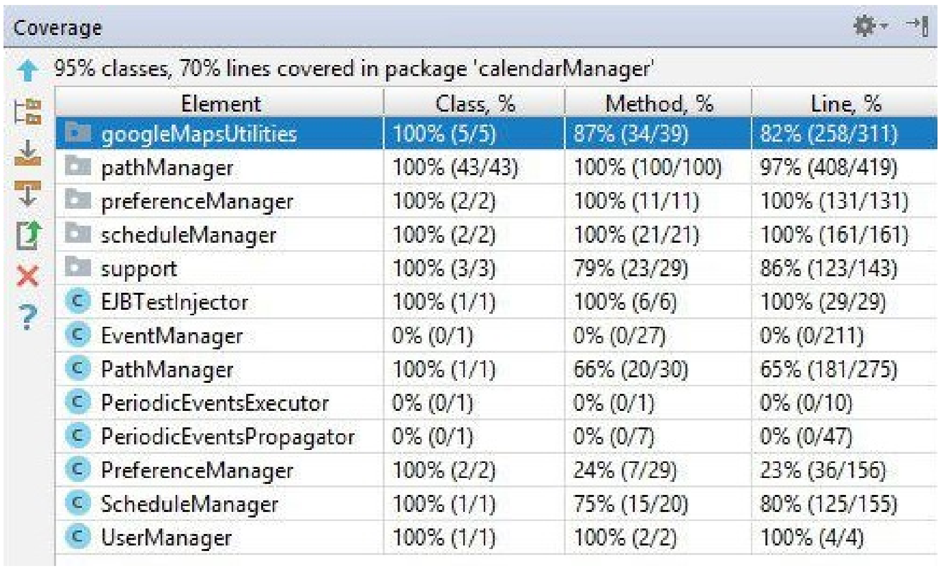
All the previous test cases are checked for the inserted event and its following one (since, when an event is inserted, the following path can change). When a following event does not exist the following path does not exist and so it has not to be checked.

We tested also *swapEvents* function, a function that forces an overlapping event into the schedule and removes from the schedule all the events in conflict with the added one.
We verified the right behavior of the function in the following cases:

- when the forced event overlaps with another one, the latter has to be removed from the schedule;
- when the forced event's path overlaps with another event, the conflicting event has to be removed from the schedule;

- when the forced event is inserted into the schedule and the following one is too close in time (and so a feasible path does not exist anymore), the following one has to be removed from the schedule;
- when the forced event is inserted into the schedule and it overlaps with a break event, but its minimum time is still guaranteed, the break event must remain into the schedule;
- when the forced event is inserted into the schedule and it overlaps with a break event, whose minimum time is no more guaranteed, the break event must be removed from the schedule.

We used Junit in order to test the beans that do not contain components difficult to be tested.


Coverage

95% classes, 70% lines covered in package 'calendarManager'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| googleMapsUtilities | 100% (5/5) | 87% (34/39) | 82% (258/311) |
| pathManager | 100% (43/43) | 100% (100/100) | 97% (408/419) |
| preferenceManager | 100% (2/2) | 100% (11/11) | 100% (131/131) |
| scheduleManager | 100% (2/2) | 100% (21/21) | 100% (161/161) |
| support | 100% (3/3) | 79% (23/29) | 86% (123/143) |
| EJBTestInjector | 100% (1/1) | 100% (6/6) | 100% (29/29) |
| EventManager | 0% (0/1) | 0% (0/27) | 0% (0/211) |
| PathManager | 100% (1/1) | 66% (20/30) | 65% (181/275) |
| PeriodicEventsExecutor | 0% (0/1) | 0% (0/1) | 0% (0/10) |
| PeriodicEventsPropagator | 0% (0/1) | 0% (0/7) | 0% (0/47) |
| PreferenceManager | 100% (2/2) | 24% (7/29) | 23% (36/156) |
| ScheduleManager | 100% (1/1) | 75% (15/20) | 80% (125/155) |
| UserManager | 100% (1/1) | 100% (2/2) | 100% (4/4) |

Figure 5.3: textitCalendarManager - code coverage

### 5.1.3 RESTful API Postman

In order to test our RESTful APIs we have performed our requests using a useful software tool: Postman. This software tool allowed us to build API requests and see their outcome, checking that:

- all invalid (due to wrong message format) requests are refused;
- all requests that try to access secured resources without being authenticated (with a token) receive an unauthorized response message;
- all requests with a right message format, but with invalid syntax, receive a bad request response message;
- all the correct requests gets replied with the correct response message.

(For additional info about our RESTful API please refer to our relative documentation link).

# Chapter 6

# INSTALLATION INSTRUCTIONS

## 6.1 Application Server and DBMS

This section contains a short guide to install properly and test the software required to run Travlendar+'s Application Server and the DBMS.
**We've already deployed in a virtual server, with a Linux OS (Ubuntu 16.04.2 LTS), our Application Server and we recommend you to use it**.
If you want to run on your computer our Application server, the following sections contain details on how to install the required software on three operating systems: Linux, Windows and macOS.
**If you want to test our application using our server, please jump to section 6.2**.
Notice that we want to provide instructions as generic as possible, so we will not provide any IDE - related instructions; feel free to use your own IDE to run both Glassfish 5.0 and MySQL 5.7, but if you never used/run these tools please follow strictly our instructions or simply use our online server.

### 6.1.1 Server Prerequisites

Check that your platform hardware meets the following prerequisites, if you want to read the full prerequisites please head to: https://javaee.github.io/glassfish/doc/5.0/release-notes.pdf, here in fact we have provided a brief summary of that document. As for MySQL, the developers do not provide any minimum requirement.

- **Required JDK Version:** GlassFish Server Open Source Edition Release 5.0 requires Oracle JDK 8 Update 144 or later;
- **Required Disk Space:** The installation size will vary depending on your configuration, but the approximate amount of disk space used by GlassFish Server 5.0 is 138 MB;
- **Required RAM memory:** It is recommended that any machine that is hosting a GlassFish Server instance has a minimum of 1 GB RAM (2 GB recommended);
- **Paths and Environment Settings for the JDK Software:** Ensure that your JDK configuration settings on all local and remote GlassFish Server hosts adhere to the guidelines listed below. Failure to adhere to these guidelines can cause various problems that may be difficult to trace.
  - **Use the JDK Binaries:** The following binary files that are used with GlassFish Server must come from the JDK software, not the Java Runtime Environment (JRE) software: *1. java*, *2. keytool*. To meet this requirement, ensure that the bin directory for the JDK software precedes the bin directory for the JRE software in your path.

  - **Set the JAVA_HOME Environment Variable:** Before performing any GlassFish Server installation or configuration procedures, set the JAVA_HOME environment variable on the GlassFish Server host machine to point to the correct Java version. Also be sure to add the JAVA_HOME/bin directory to the PATH variable for your environment.

### 6.1.2 Windows 10

#### Install and setup MySQL 5.7

- Please follow the guide provided in beep course's web page, in slides folder: "JEE Tools Instructions for Lab.pdf", (only section 3.3) in order to proper install MySQL;
- Open a terminal and navigate to `'yourMySQLInstallationFolder'\bin` and run the command 'mysqld.exe' in order to start MySQL;
- Open another terminal and navigate to `'yourMySQLInstallationFolder'\bin` and run the command 'mysql -u root -p' in order to access MySQL's command line console;
- MySQL's command line console will ask for a password, just press 'enter';
- In MySQL's command line console's window type 'create database mps_travlendar;' and press 'enter' in order to create the database needed by the application server;
- Before starting to use the Android app please follow the provided steps to install and setup Glassfish.

#### Install and setup Glassfish 5.0

- Download GlassFish Server 5.0 ZIP file from here:
  http://download.oracle.com/glassfish/5.0/release/glassfish-5.0.zip;
- Head to the directory where you downloaded the ZIP file;
- Extract the file into the directory in which you want to install your GlassFish server;
- Head to `'yourGlassfishMainFolder'\bin` and double click on the asadmin.bat file in order to open GlassFish' command line terminal;
- In the opened terminal type 'start-domain' and press enter in order to start GlassFish server;
- Head to `'yourGlassfishMainFolder'\glassfish\domains\domain1\autodeploy` and copy in this folder the ApplicationServerArchive.war file: link;
- Wait a few seconds and then the application server will be deployed.

NB. To stop GlassFish type 'stop-domain domain1' in the same terminal you've started it.

### 6.1.3 Linux

#### Install and setup MySQL 5.7

We have encountered several problems following the guide provided by our instructors about MySQL on a linux OS, so we will provide some alternatives to be sure that you succeed to install it; you can either:

- follow the guide provided in beep course's web page, in slides folder: "JEE Tools Instructions for Lab.pdf", (only section 2.3) in order to proper install MySQL, or
- follow the official guide provided on MySQL official website: https://dev.mysql.com/doc/refman/5.7/en/linux-installation.html.

Once you have successfully installed MySQL you have to:

- Open a terminal and run the command '/etc/init.d/mysql start' (or 'service mysql start', or 'mysql.server start') in order to start MySQL;
- Open a terminal and run the command 'mysql -u root -p' in order to access to MySQL's command line console;
- MySQL's command line console will ask for a password, just press 'enter';
- In MySQL's command line console's window type 'create database mps_travlendar;' and press 'enter' in order to create the database needed by the application server;
- Before starting to use the Android app, please follow the provided steps to install and setup Glassfish.

#### Install and setup Glassfish 5.0

- Download GlassFish Server 5.0 ZIP file from here: http://download.oracle.com/glassfish/5.0/release/glassfish-5.0.zip;
- Change to the directory where you downloaded the ZIP file;
- Extract the file into the directory in which you want to install your GlassFish server;
- Open a terminal and navigate to `'yourGlassfishMainFolder'\bin` and run the command './asadmin start-domain' in order to start GlassFish server;
- Head to `'yourGlassfishMainFolder'\glassfish\domains\domain1\autodeploy` and copy in this folder the ApplicationServerArchive.war file: link;
- Wait a few seconds and than the application server will be deployed.

NB. To stop GlassFish open a terminal, navigate to `'yourGlassfishMainFolder'\bin` and run the command './asadmin stop-domain domain1'.

### 6.1.4 macOS

#### Install and setup MySQL 5.7

- Please follow the guide provided in beep course's web page, in slides folder: "JEE Tools Instructions for Lab.pdf", (only section 4.3) in order to proper install MySQL;
- Open your System Preferences, click on MySQL icon and click on "Start MySQL server" or open a terminal and run the command 'mysql.server start' in order to start MySQL;
- Open a terminal and run the command 'mysql -u root -p' in order to access to MySQL's command line console;
- MySQL's command line console will ask for a password, just press 'enter';

- In the MySQL's command line console's window type 'create database mps_travlendar;' and press 'enter' in order to create the database needed by the application server;
- Before starting to use the Android app please follow the provided steps to install and setup Glassfish.

**Install and setup Glassfish 5.0**

- Download GlassFish Server 5.0 ZIP file from here: http://download.oracle.com/glassfish/5.0/release/glassfish-5.0.zip;
- Change to the directory where you downloaded the ZIP file;
- Extract the file into the directory in which you want to install your GlassFish server;
- Open a terminal and navigate to `'yourGlassfishMainFolder'\bin` and run the command './asadmin start-domain' in order to start GlassFish server;
- Head to `'yourGlassfishMainFolder'\glassfish\domains\domain1\autodeploy` and copy in this folder the ApplicationServerArchive.war file: link;
- Wait a few seconds and than the application server will be deployed.

NB. To stop GlassFish open a terminal, navigate to `'yourGlassfishMainFolder'\bin` and run the command './asadmin stop-domain domain1'.

## 6.2 Android App

This section contains a short guide to proper install the software required to run Travlendar+'s Android App.

### 6.2.1 Instructions to run the App on your computer

- Open your browser and head to https://developer.android.com/studio/index.html;
- Click on the 'Download Android Studio' green button;
- Accept the license agreement (check the box that states 'I have read and agree with the above terms and conditions');
- Click on the 'Download Android Studio for <your OS name>' blue button (NB <your OS name> represent the name of your Operating System);
- The download process should automatically start and your browser should redirect you on the 'Install Android Studio' web page. If it does not so head to: https://developer.android.com/studio/install.html;
- Follow the instructions relative to your OS, and **make sure that 'Android virtual device' is selected for installation**.

**If you're using our deployed server (no local server - RECOMMENDED):**

- At the end of the installation process Android Studio shows a window named 'Welcome to Android Studio'. Click on 'Profile or debug APK';
- Select Travlendar+'s APK location: link;
- If the window 'Tips of the day' opens up, click on Close;
- On the menu bar click on tools ->Android ->ADV manager;
- Click on the 'Create Virtual device...' button;
- Select 'Phone as category' and then 'Pixel 2 XL';
- Click Next;
- Select Oreo system image (Android 8.0) in 'Recommended' tab;
- Click on the download link to the right of 'Oreo' in order to download the selected Android OS and wait until the download ends;
- Click Next;
- Click Finish;
- Close the 'Android Virtual device Manager' window;
- On the menu bar click on Run ->Run 'travlendarPlus';
- Select 'Pixel 2 XL API 26' virtual device and click 'Ok';
- Wait until your Android virtual device starts and installs our App;
- Make sure that the Android virtual device's and your computer's current time are the same, if not go to Settings app, click on system ->Date & Time, deselect 'Automatic time zone' and Select manually your current time zone. Now the two clocks should display the same current time;
- Enjoy Travlendar+'s Android App.

**If you're not using our deployed server (local server):**

- While Android Studio installation's goes on, clone our GitHub repo (link) and take note of the location where the repo will be cloned at;
- At the end of the installation process Android Studio will show a window named 'Welcome to Android Studio', click on 'Open an existing Android studio project';
- Select `'cloneLocation'\Implementation\AndroidApp\Travlendar` and then click Ok;
- If 'Tips of the day' window opens up click on Close;
- In the 'Android tab' (top right of the window) Select `app\java\it\polimi\travlendarplus\retrofit\` and open ServiceGenerator.java;
- Change in the constant BASE_URL the address '151.236.60.56' with 'localhost' or with the IP address of the computer where you have deployed our server;

- On the menu bar click on tools ->Android ->ADV manager;
- Click on 'Create Virtual device...' button;
- Select 'Phone as category' and then 'Pixel 2 XL';
- Click Next;
- Select Oreo system image (Android 8.0) in 'Recommended' tab;
- Click on the download link to the right of 'Oreo' in order to download the selected android OS and wait until the download ends;
- Click Next;
- Click Finish;
- Close the 'Android Virtual device Manager' window;
- On the menu bar click on Run ->Run 'travlendarPlus';
- Select 'Pixel 2 XL API 26' virtual device and click 'Ok';
- Wait until your Android virtual device starts and installs our App;
- Make sure that the Android virtual device's and your computer's current time are the same, if not go to Settings app, click on system ->Date & Time, deselect 'Automatic time zone' and Select manually your current time zone. Now the two clocks should display the same current time;
- Enjoy Travlendar+'s Android App.

### 6.2.2 Instructions to run the App on your Android device

**(from Android version 6.0)**

- Download the APK of Travlendar+ from here: link;
- Verify that the Android version of your Android device is greater or equal to 6.0 version. You can find out the Android version that is currently running on you device in Settings ->About phone ->Android version;
- Open the downloaded APK file, then click on "Install" button;
- Register your account with an email address in a valid format;
- Enjoy Travlendar+'s Android App.

Remember that our application is supported from the 6.0 Android version; however, it has been developed and mainly tested on a virtual device with an 8.0 version. For this reason, **we do not ensure that the use of Travlendar+ on previous than 8.0 Android versions is error free**.

# Chapter 7

# EFFORT SPENT

## 7.1 Pietro Melzi

| Date | Task | Hours of work |
|---|---|---|
| 27/11/17 | Group session - Project initialization | 1 |
| 28/11/17 | First entity classes written | 2 |
| 29/11/17 | Add of entities, study of GMaps solutions | 4 |
| 30/11/17 | Group session - Package organization | 1 |
| 01/12/17 | Simulation of GMaps calls and considerations on the solution to adopt | 3 |
| 02/12/17 | Group session with Salvadore - Discussion on GMaps calls, configuration of the server, restful simulation | 8 |
| 03/12/17 | Authentication and security, name binding and injection | 6 |
| 04/12/17 | Schedule manager functions | 7 |
| 05/12/17 | GMaps functions and reading of the parameters in JSON response | 6 |
| 06/12/17 | Group session with Salvadore - Configuration of server on Register domain, fixes on reading of JSON | 7 |
| 07/12/17 | JSON part completely working, travel functions and overlapping functions | 8 |
| 08/12/17 | Corrections on schedule functions, first version of swap function | 6 |
| 09/12/17 | Junit and corrections on schedule manager functions | 7 |
| 10/12/17 | First-path function, path combination class, swap correction | 5 |
| 11/12/17 | Fixes on GMaps calls, calculate path function working | 6 |
| 12/12/17 | Test and fixes on path calculation, with first-last travel, easy swap examples and application of preferences | 7 |
| 13/12/17 | Fixes on the code and setting of departure location based on user's preferences | 6 |
| 14/12/17 | Other corrections on calculate paths function | 4 |
| 15/12/17 | Group session with Salvadore - Exceptions, corrections of errors, travel ASAP or ALAP, test | 8 |
| 16/12/17 | Group session - Fix errors on swap function, study of Mockito, schedule holder tested | 11 |
| 17/12/17 | Test of gMaps direction APIs and of JSON reader, fixes on the calculation of paths with long by foot distance | 6 |
| 18/12/17 | Mockito tests on path manager functions | 6 |

| | | |
|---|---|---|
| 19/12/17 | Other Mockito tests, fix on the calculation of paths with very short by foot distance | 6 |
| 20/12/17 | Group session with Salvadore - General fixes and improvement on swap function | 6 |
| 21/12/17 | Swap test in all cases | 5 |
| 22/12/17 | Group session - Time fixed in public travels with by foot components, first part of the break event-swap function | 7 |
| 23/12/17 | Group session - Break event-swap function completed, fixed bugs on long by foot paths and on calls with time in the past | 9 |
| 24/12/17 | Organization of test classes and creation of test utilities, preference manager tested | 6 |
| 27/12/17 | Group session with Salvadore - Fixes on path calculation in particular conditions and java doc, test of functionalities offered by the app | 9 |
| 28/12/17 | Group session with Salvadore - Test on ticket functionalities on the app, section 3 of document | 9 |
| 29/12/17 | Group session with Salvadore - Global revision of the project, test section in the document, debug and corrections on the app | 13 |
| 30/12/17 | Group session with Salvadore - Corrections of last errors, revision of the document and delivery | 10 |

## 7.2 Alessandro Pina

| Date | Task | Hours of work |
|---|---|---|
| 11/25 | Android project creation | 4 |
| 11/26 | Login/registration layouts | 4 |
| 11/27 | Main activities layout | 4 |
| 11/29 | Preferences activity layout | 3 |
| 11/30 | Activities | 6 |
| 12/01 | Local DB initialization | 3 |
| 12/02 | Local DB issues | 4 |
| 12/03 | Local DB queries | 3 |
| 12/04 | Local DB working with UI | 6 |
| 12/05 | Layout improvements | 3 |
| 12/06 | UI improvements | 5 |
| 12/07 | Local DB issues | 3 |
| 12/08 | Dragging events implementation | 6 |
| 12/09 | Registration to server | 5 |
| 12/10 | Login to server | 6 |
| 12/11 | Locations API implementation | 6 |
| 12/12 | Authorization issues | 5 |
| 12/13 | Preferences requests | 5 |
| 12/14 | Preferences issues | 4 |
| 12/15 | Retrofit implementation | 4 |
| 12/16 | Retrofit implementation | 4 |
| 12/17 | Preferences add and edit | 5 |
| 12/18 | Preferences working with server | 6 |
| 12/19 | Events get and add | 8 |
| 12/20 | Events displaying on UI | 8 |

| | | |
|---|---|---|
| 12/21 | Documentation and fixes | 5 |
| 12/22 | Schedule travels | 8 |
| 12/23 | Tickets, events swapping | 8 |
| 12/24 | Add tickets, bug fixes | 8 |
| 12/26 | Bug fixes | 4 |
| 12/27 | Travel components | 10 |
| 12/28 | Linking tickets, bug fixes | 4 |
| 12/29 | DB testing, ITD | 6 |
| 12/30 | Bug fixes, locations on map | 2 |

## 7.3 Matteo Salvadore

| Date | Task | Hours of work |
|---|---|---|
| 27/11/17 | Group session - Project initialization | 1 |
| 28/11/17 | JPA entities pt. 1 | 5 |
| 29/11/17 | JPA entities pt. 2 and JPA mapping | 5 |
| 30/11/17 | Group session - package organization | 1 |
| 1/12/17 | JPA entities pt. 3 and JPA mapping | 5 |
| 2/12/17 | Group session with Melzi - Maven added into the project, introduced Unix time, setup of JAX-RS API, discovered conflict between GlassFish 5.0 and Yoda time | 10 |
| 3/12/17 | Google Maps's library fork to try solving Yoda time conflict with Glassfish - Failed, JPA save methods | 8 |
| 4/12/17 | JPA save and remove methods fully working, introduced messages and exception packages, setup of user authentication | 6 |
| 5/12/17 | Authentication methods fully working, DataBase with lazy fetch settings, token generation algorithm, RSA algorithm | 6 |
| 6/12/17 | Group session with Melzi - Linux remote server setup, deploy of a first release of our project, troubleshooting of connectivity problems (ex. PoliMi's WiFI lock some ports...) and remote DB integration | 6 |
| 7/12/17 | Event manager, its relative RESTful interface, initialization of preference manager | 9 |
| 7/12/17 | Server Deploy and Postman project folder with RESTful requests format for Pina | 2 |
| 8/12/17 | PreferenceManager- event managers, bugfixes, postman preferencemanager folder | 8 |
| 9/12/17 | Methods that performs message fields consistency in event and preference manager, path and schedule RESTful methods, Arquillian setup | 10 |
| 10/12/17 | Setup of Arquillian DB testing - unsuccessful, wasted time | 4 |
| 10/12/17 | Email sender, periodic events' propagation, public key timer | 5 |
| 11/12/17 | Periodic event's propagation, JavaDoc documentation of RESTful methods, optimization of authentication methods, server deploy | 7 |
| 12/12/17 | Arquillian testing - other configuration problems | 7 |
| 13/12/17 | Arquillian JPA testing, server redeploy after several fixes | 5 |
| 14/12/17 | JPA testing | 6 |
| 15/12/17 | Group session with Melzi - JPA testing completed, project revision, Greenwich fix | 5 |

| | | |
|---|---|---|
| 15/12/17 | Server redeploy, periodic events' propagation, event Manager fixed, postman event's RESTful folder for Pina | 4 |
| 16/12/17 | Group session with Melzi - EventManager, application server reorder | 6 |
| 16/12/17 | Group session - Arrange trips RESTful methods | 3 |
| 16/12/17 | Trip Manager backbone | 4 |
| 17/12/17 | Trip manager - ticket functionalities completed | 4 |
| 17/12/17 | Test fixed due to entity modifications, troubleshooting of event and preference managers, now event can be modified, changed responses format of preference manager after Pina request, server re-deployed after fixes, encryption of passwords | 5 |
| 18/12/17 | Fixes on event propagation and RESTful messages, server redeploy, asynchronous thread for event propagation | 6 |
| 19/12/17 | JavaDoc documentation, minor fixes, server redeploy | 6 |
| 20/12/17 | Group session with Melzi: Gmaps Exceptions' propagation, break event's null pointer exception fixed, server redeploy (many times) | 5 |
| 20/12/17 | JavaDoc documentation, redefined string messages as constants, RSA encryption, Gmaps Exceptions propagation | 6 |
| 21/12/17 | Gmaps Exceptions propagation, checks on ticket-travel selection, cleaned maven dependencies, improved delete event (propagation of the next event's travel) | 5.30 |
| 22/12/17 | Group session - RSA, changed ticket list response, only one simultaneously device per user check, changed error list response, debug and fixes, several server deploy | 11 |
| 23/12/17 | Group session - RESTful swap method is now available also for break events, period ticket response message fixed, RSA, several server deploy, merge into master of the Applicaton-Server branch | 9 |
| 23/12/17 | debug and then swap ids fixed | 1 |
| 24/12/17 | Wasted time installing Android Studio (Windows firewall related problems) | 4 |
| 24/12/17 | Android app debug | 3 |
| 26/12/17 | App and server debug and therefore some fixes, installation instructions for both server and app | 7 |
| 27/12/17 | Group session with Melzi - some fixes, DD corrections, server redeploy, ITD document | 9 |
| 27/12/17 | Application server JavaDoc published on GitHub pages, ITD: application server code structure | 2 |
| 28/12/17 | Group session with Melzi - some fixes, Android app debug, ITD : Android code structure, installation instructions corrections, implemented requirements in android, RESTful documentation on GitHub | 9 |
| 28/12/17 | Android app debug, ITD: Purpose, scope and general corrections | 2 |
| 29/12/17 | Group session with Melzi - Android app debug,white spinner fixed, wrong message format fixed, ITD: Arquillian framework and test. Apk build | 13 |

| | | |
|---|---|---|
| 30/12/17 | Group session with Melzi - Android app debug, ITD: last fixes, delivery | 13 |

# Chapter 8

# REFERENCES

## 8.1 Bibliography

(I) GlassFish Server Open Source Edition - Release Notes - Release 5.0

(II) GlassFish Server Open Source Edition - Installation Guide - Release 5.0

(III) GlassFish Server Open Source Edition - Installation Guide - Release 5.0

(IV) Document provided in beep course web page in slides folder: "JEE Tools Instructions for Lab";

## 8.2 Software used

This document is written in LaTex, a markup language used for text editing, and includes sections realized with different software tools. Here we show the list of the software used:

- Texmaker 5.0.2 (compiled with Qt 5.8.0) for writing the document;
- draw.io for diagrams.

In order to implement and test our project we have used the following software tools:

- IntelliJ IDEA Ultimate Edition 3.2 to develop our application server ;
- Android Studio 3.0.1 to develop our Android app;
- Postman for Windows 5.5.0 to simulate RESTful requests to our application server;
- GlassFish 5.0 as JEE engine to run our application server;
- MySQL Community Server 5.7.20 as DBMS system, connected to our application server;
- Putty (0.70), an SSH and telnet client we have used to access to our server's terminal in remote.
- Google Chrome 63.0, used to check the state of the local DB in the Android App during development.