POLITECNICO
MILANO 1863

Travlendar+
Design Document

Fumagalli Paolo, Grotti Pietro, Gullo Marco

November 25, 2017

Contents

1 Introduction

1.1 Purpose

This Design Document wants to continue the analysis of the application called Travlendar+, which was started in the Requirement Analysis and Specification Document. The application wants to help people organize their daily schedules by providing suggestions on the best path to follow, based on their preferences. This document wants to:

- Identify the high level architecture of the application, along with its components and interfaces.
- Define the most important algorithms which will be used during the implementation part.
- Give an initial mock-up of the interface to show how Travlendar+ will look like after the implementation is done.
- Associate the requirements from the RASD document to the specifics defined below in this document.
- Study the best way to implement the different components of the application by giving an order in which they will be implemented and tested.

1.2 Scope

Travlendar+ is an application which will be able to manage daily appointments of the users and assist them by creating a specific course around the city which will identify the best mobility option to move from one appointment to the other. It will consider all public transportation options (train, metro, bus, etc.), car and bike sharing systems, the eventuality of a private vehicle and the weather conditions as well, to avoid having the user bike in harsh weather conditions or when it is too hot. It will also be possible to buy tickets and locate cars and bikes directly through the application. Users will have to create meetings specifying the location, the date and the time and the application will automatically calculate the suggested course and travel time, if it is impossible to reach a certain meeting in time the application will send a warning to the user. It will also feature the possibility of selecting a time span in which it will have to save some time for a break in which the user can have lunch.

1.3 Definitions, Acronyms, Abbreivations

1.3.1 Definitions

1.3.2 Abbreviations

G* : Specific goal
R* : Specific functional requirements
D* : Specific domain assumption
App : Application

1.3.3 Acronyms

ETA : Estimated Time of Arrival
API : Application Programming Interface
PTS : Public Transportation System
CSS : Car-Sharing System
BSS : Bike-Sharing System
RASD: Requirement Analysis and Specifications Document
DD: Design Document
REST: REpresentational State Transfer
CRUD: Create, Read, Update, Delete
CCB: Cluster Computing Base
ODBC: Open DataBase Connectivity

1.4 Revision History

Version 1.0

1.5 Reference Documents

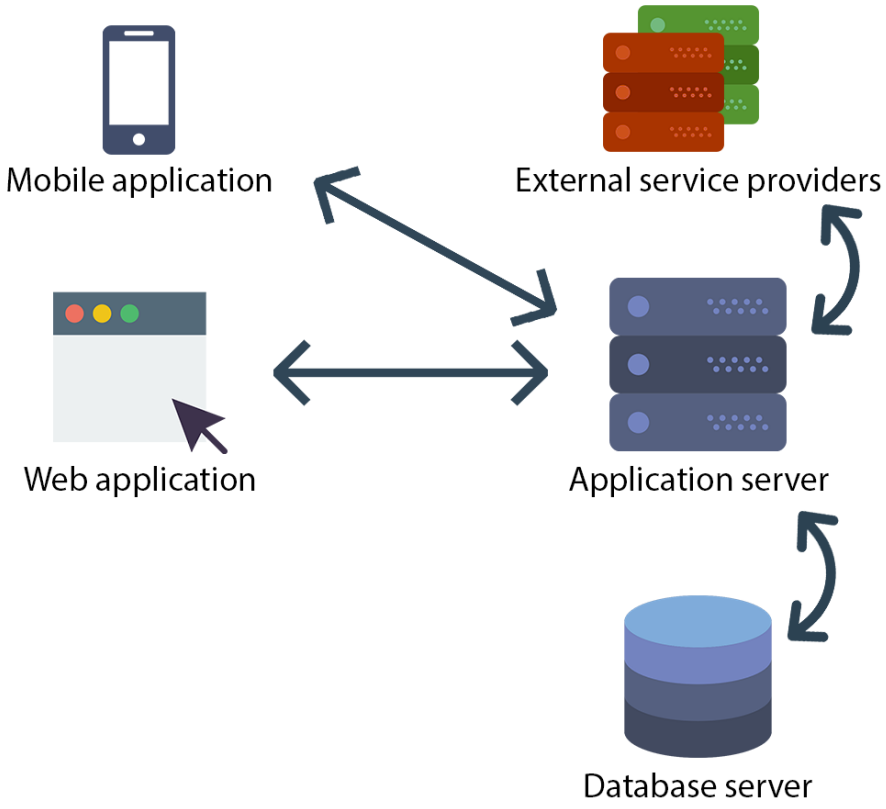
1.6 Document Structure

The structure of the document follows the standard IEEE and is divided into six chapters.
After a brief introduction in chapter 1, chapter 2 analyses the components of the system and how they interact with each other. The final part of this chapter will give suggestions to the developing team on which design patterns should be used.
The third part of the document focuses on the most important algorithms that will be used during the development part, which will be improved and optimized while developing the application. The fourth chapter focuses on mockups of the user interface and contains UX and BCE diagrams of the application and website.
The fifth part will link the RASD to the DD by showing how the requirements

defined in the first document are mapped in the design elements studied in this document.
The last part will provide insight on how the application will be developed and tested, once this document is ready and sent to developing team.

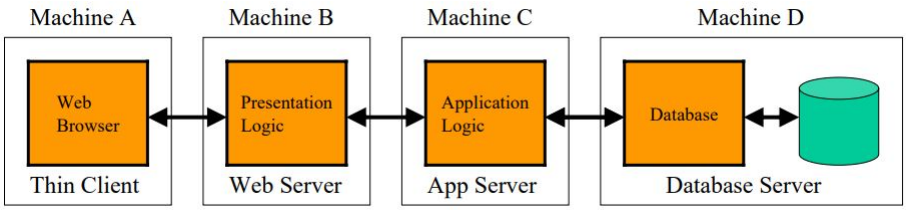
2 Architectural Design

2.1 Overview: High-level components and interactions



Architecture's Ideal Concept

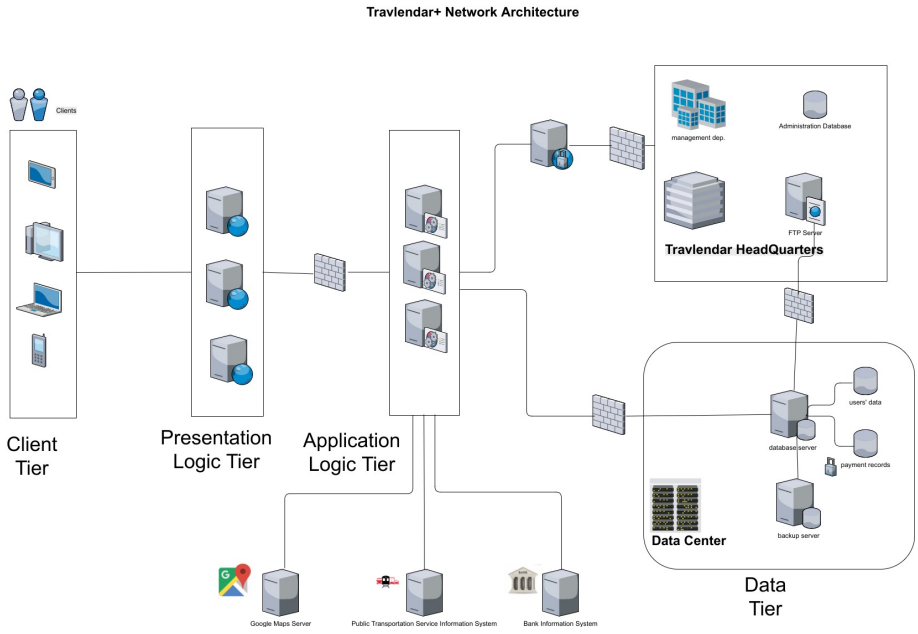
Even if Travlendar+ is accessible both with mobile and web browser, it was clear from the beginning that, for this projects purposes , the applications strength will lie on the mobile version. Since the threetier architecture lacks scalability (in fact, it was designed in an era where the idea of elasticity and rapid scaling did not broadly exist) a four-tier one was chosen. Four-tier was recently introduced for mobile applications, so it is the best fit. This way it is also guaranteed that different types of client can share the same application logic.



Four Tier Architecture Paradigm Example

The layers are:

- **Client Tier:** Clients interfaces and local part of the application.
- **Presentation logic Tier:** Web Servers, responsible of delivery, the so-called communication between client and application servers. Here is the REST interface. In this tier the different client types are masked to the Application Servers.
- **Application logic Tier:** Cluster computing base where the several operations (in a CRUD interface) that can be done through Travlendar+ are processed, data are acquired from the internal database and it is handled the interaction with external providers and secure third-party APIs. Externals provide access to services (e.g.: ticket buying) or data (e.g.: shortest path).
- **Data Tier:** The database server stores all the clients data (calendar, settings, etc); information can be divided into users data and payment records (this database has extra security).



Physical Architecture

The top-right section of the picture above shows how Administrators will interact with the different layers: this part will not be treated in this document (it was never mentioned in RASD) and will be implemented with an ERP software the company is willing to buy from COTS market.

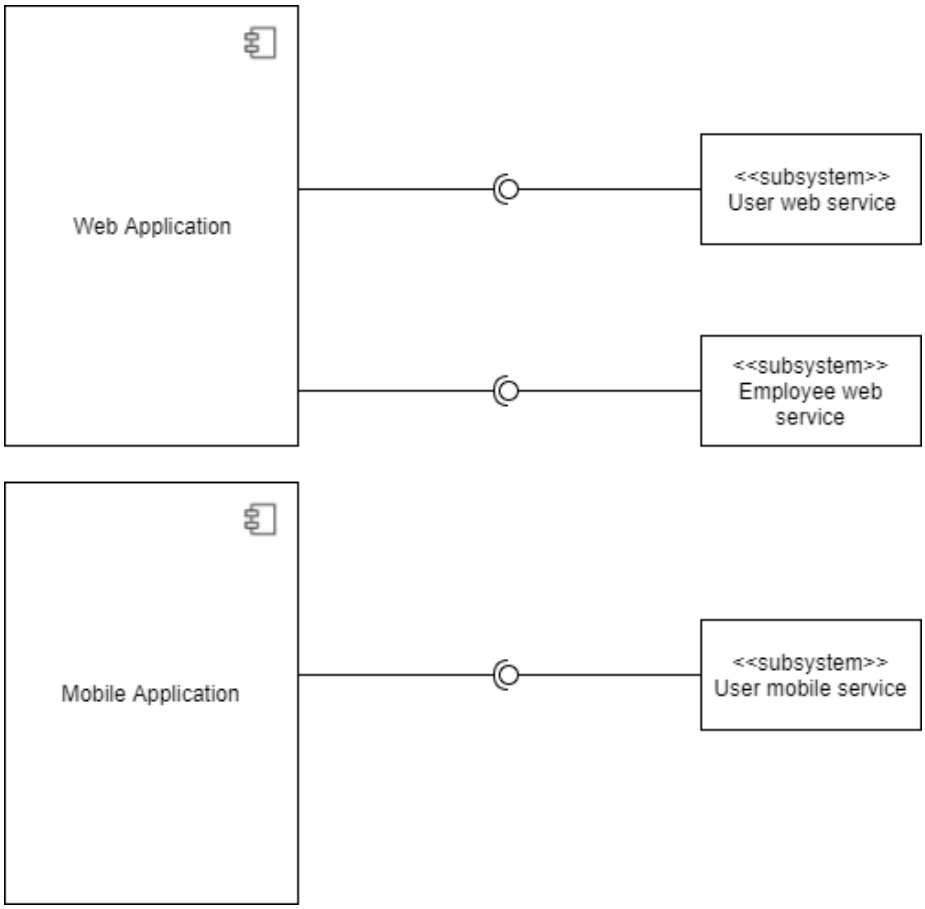
2.2 Component View

2.2.1 High Level Component Diagram

The following diagram shows the most important components of the system and which are the interfaces used to communicate and interact between modules.
The client side is made of the User Application Service and Employee Web Service, which are related to their counterparts placed on the server, and will interact with these server parts.
The server side is divided into two parts:

- User Application Service
- Employee Web Service

The user web and mobile services are almost identical, so we will discuss only one. The Employee Web Service was not discussed in the RASD and is not relevant for the purpose of the application, so we will not discuss it further.



High Level Architecture

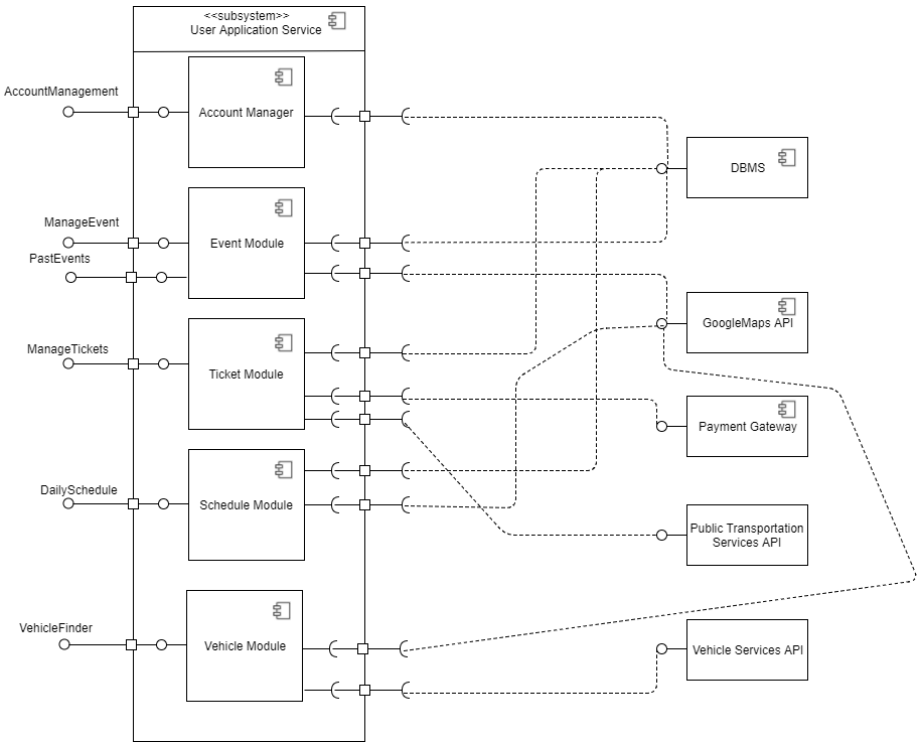
2.2.2 User Service Application

This subsystem is composed of five different modules which will provide to the user interfaces to: account management, event management, past events, manage and acquisition of tickets, daily schedule and the vehicle finder.

- **Account Manager:** this module needs to communicate with the DBMS to allow the user to change his account information and preferences.
- **Event Module:** this module needs to interact with the maps to retrieve the information of the location of the event and later save it on the database, so it needs to communicate with the DBMS.
- **Ticket Module:** this module will have to communicate with three different systems, first of all the tickets bought will have to come directly from the Public Transportation System, so it will need to interact with

the PTS. To pay for the ticket it will need to contact a Payment Gateway, and later save the information on the accounts information, so il will need to interact with the DBMS.

- **Schedule Module:** this module has to retrieve information about the events and the public transportation schedules from the DB through the DBMS and then calculate the movements through Google Maps API.
- **Vehicle Module:** this module simply needs to find the vehicles from their respective systems and show them to the user, which means that will have to interact both with Google Maps and the Vehicle Services.



User Application Service Subsystem

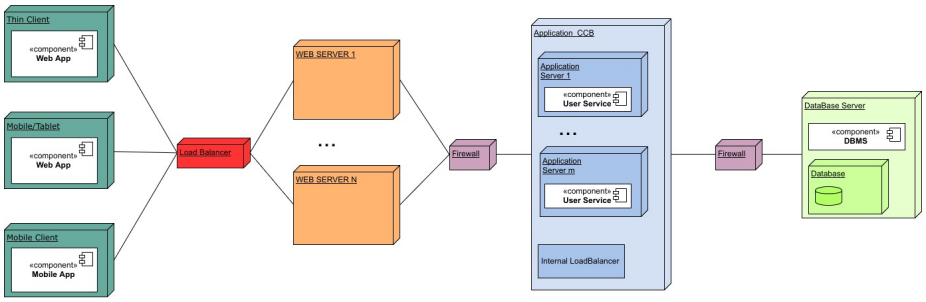
2.3 Deployment View

In this section we now match all components (section 2.2) with Travlendar+ hardware architecture (a brief overview was given in section A). If we take a look back to the diagram in Section A, the architecture is compliant with the Java EE model:

- Client Presentation Tier $\longleftrightarrow$ Client Tier
- Web Tier $\longleftrightarrow$ Presentation Logic Tier
- Business Tier $\longleftrightarrow$ Application Logic Tier
- Data Tier $\longleftrightarrow$ Data Tier

Therefore the application is very likely to be developed in Java and in a JavaEE environment (NetBeans 8.2 or Eclipse Oxygen)

Since Travlendar+ aims to reach the highest number of users, the system must handle a future high load of connections. For this purpose there is going to be more than one Apache HTTP server in the web tier and a cluster base in business tier. This multiplicity of machines will need load balancers to exploit their computational power in handling connections and executing parallel tasks in application layer. Data are stored in an external and independent Relational Database (MS SQL server), as shown in section 2.1.



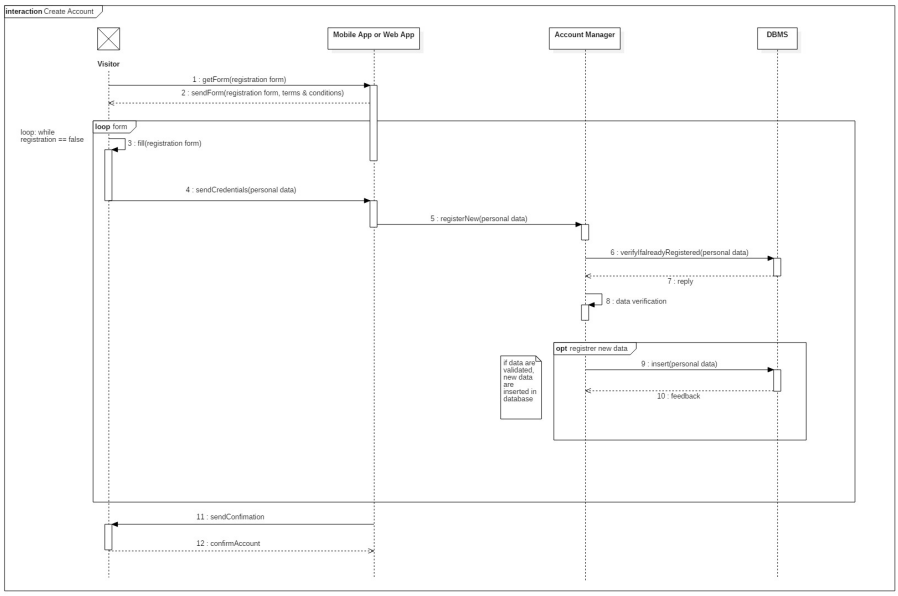
Deployment

The web and application servers will be implemented starting from open source Java EE Apache Servers code. Moreover, to allow a safer client-server communication, HTTPS encryption will be enabled on web servers. Interaction between Business and Data tier will be through a ODBC.

2.4 Runtime View

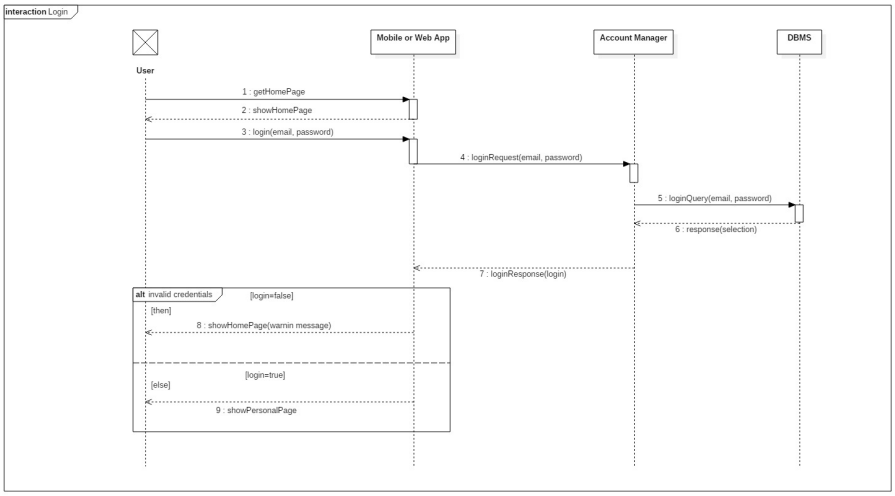
In this section all Sequence Diagrams previously showed in the RASD will be proposed in more details. In the previous document, Travlendar+s system was modeled as one entity (one lifeline in sequence diagrams); now it will be decomposed in the main subsystems and modules. The following diagrams are to give a high level snapshot of various components interactions, in cases of interest.

2.4.1 Create Account



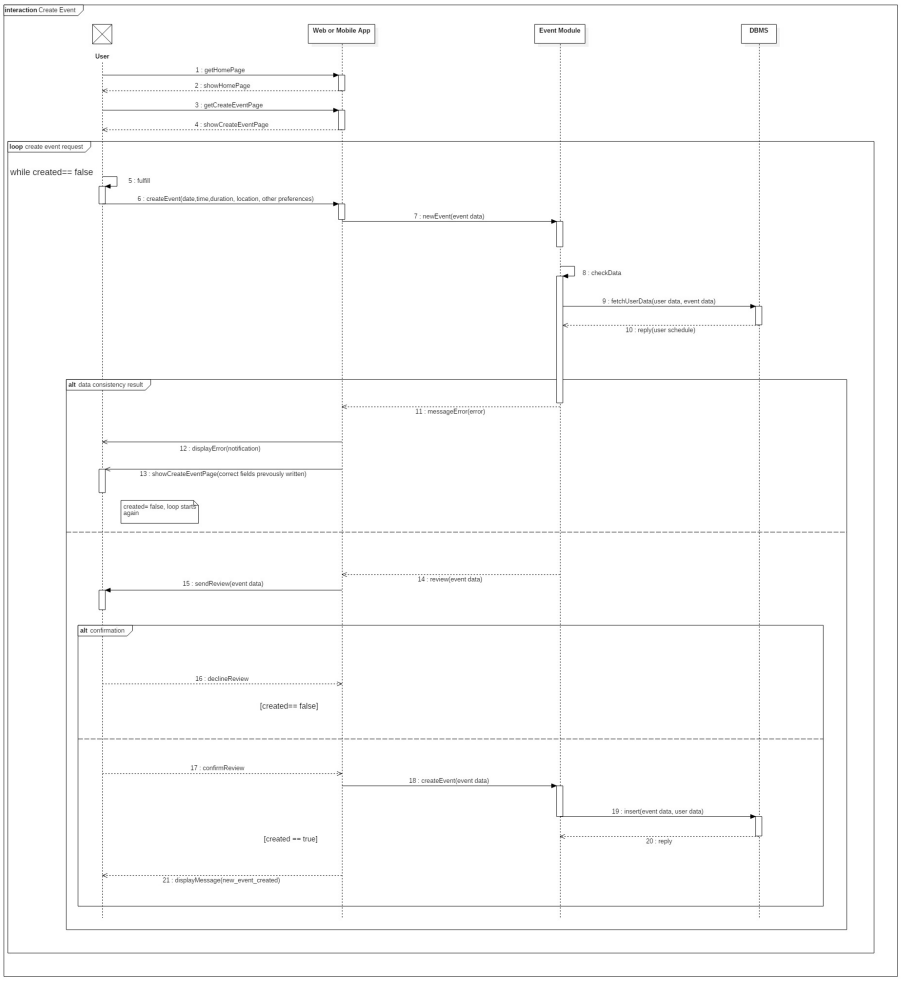
Create Account

2.4.2 Login



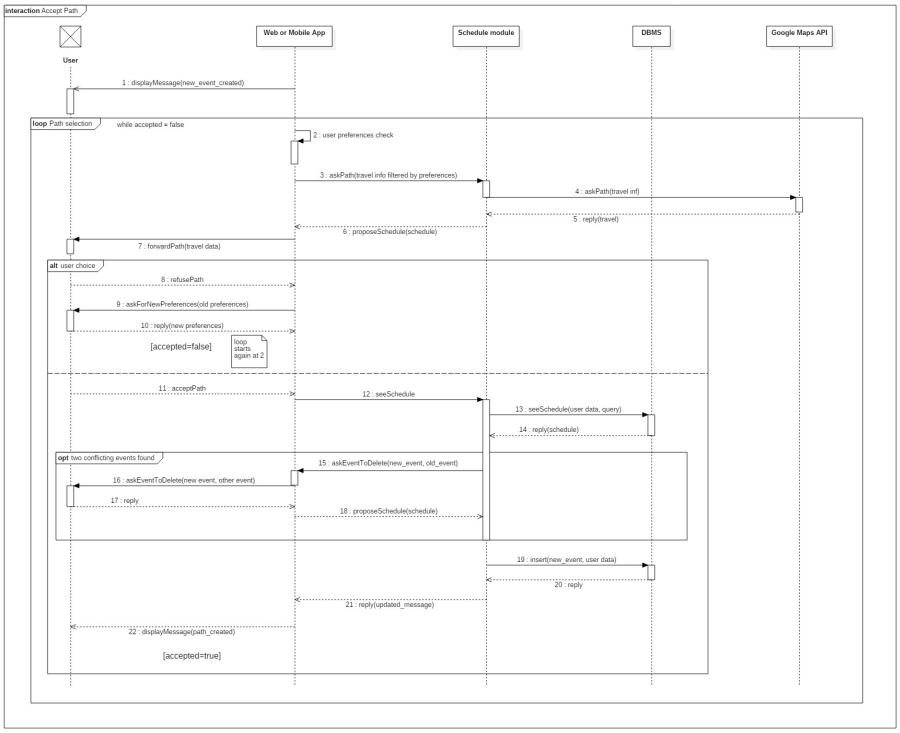
Login

2.4.3 Create Event



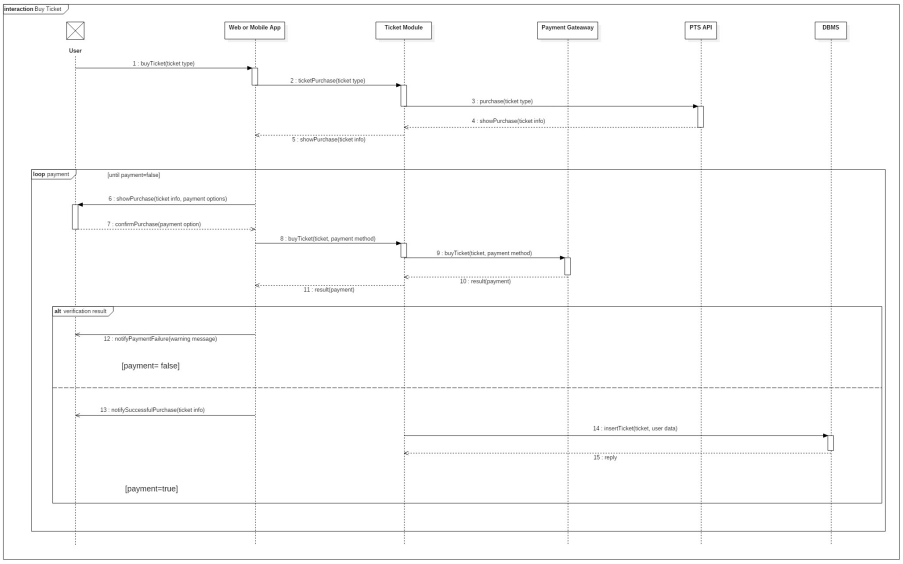
Create Event

2.4.4 Accept Path



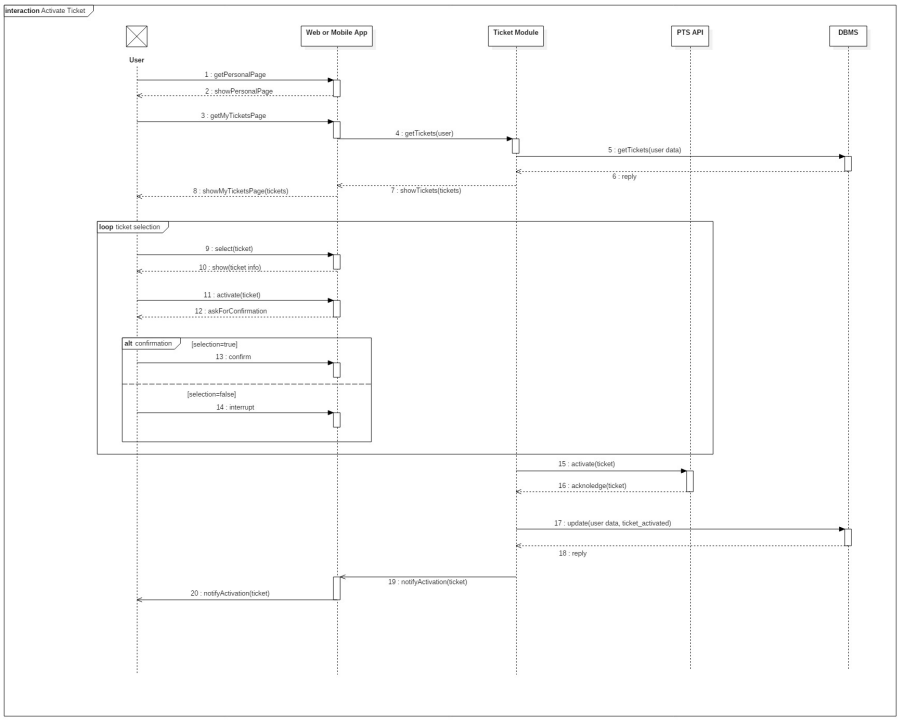
Accept Path

2.4.5 Buy Ticket



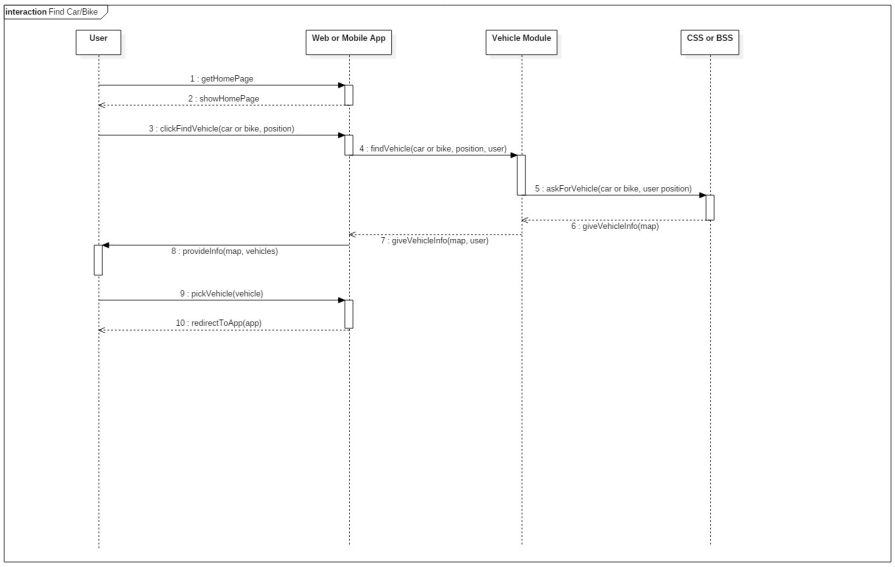
Buy Ticket

2.4.6 Activate Ticket



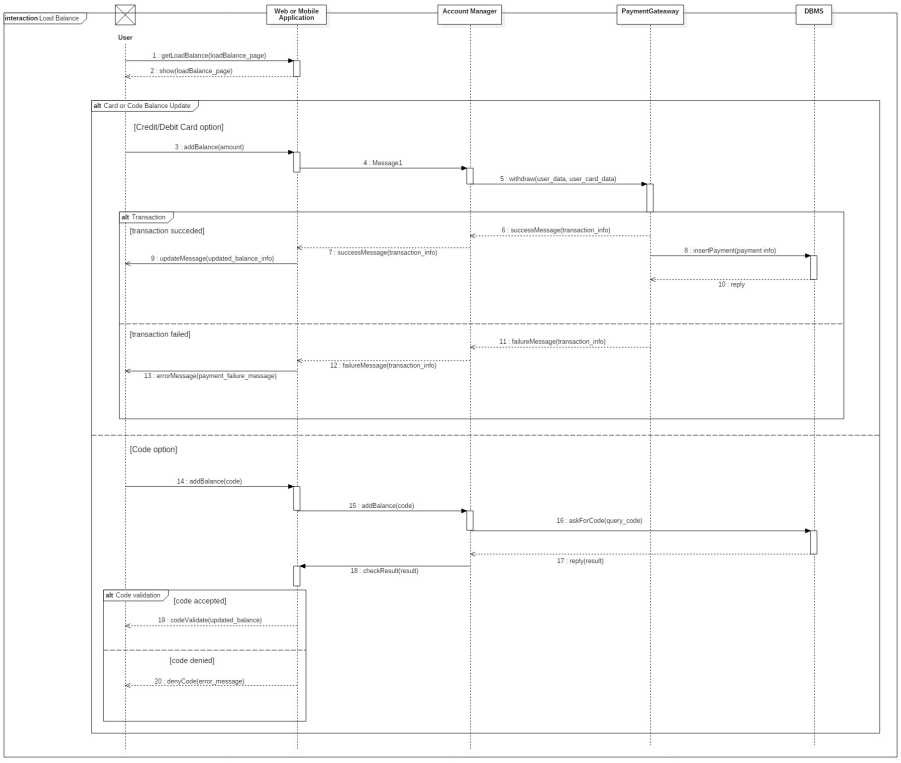
Activate Ticket

2.4.7 Find Vehicle



Find Vehicle

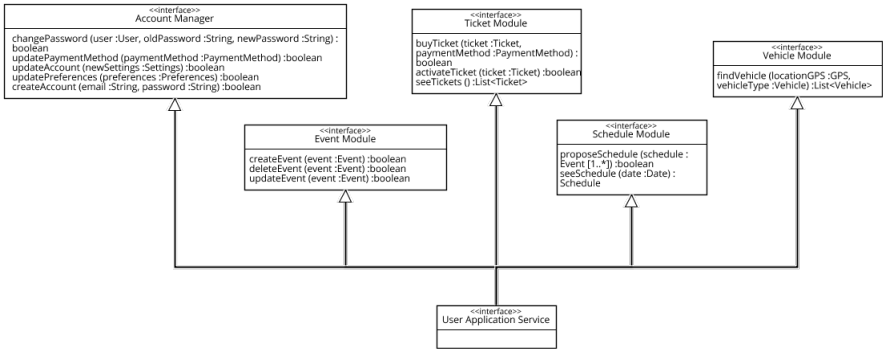
2.4.8 Load Balance



Load Balance

2.5 Component Interfaces

The diagram below shows the main methods that belong to the User Application Server. This subsystem is composed of five modules, which, through their methods, provide the client with the functionalities that the users need to use Travlendar+, such as creating an account, creating an event, buying tickets, etc.



Component Interfaces

2.6 Selected Architectural Styles and Patterns

The following patterns are recommended for the implementation of the project:

- **Model-View-Controller:** this pattern divides the software into three different components, dividing how the information is shown to the user from how it is represented in the system.
- **Proxy Pattern:** through this pattern we can use an object as an interface for another object.
- **Factory Pattern:** this pattern allows to create objects leaving the process of creation to subclasses.
- **Visitor Pattern:** this pattern is really useful to change the executing algorithm of a class, depending on which class is visiting the object.
- **Observer Pattern:** it is necessary to implement the MVC Pattern.
- **State Pattern:** this pattern allows a class to change its behaviour based on its current state.

During the implementation we will update this list citing any pattern that we will use and are not in the paragraph above.

2.7 Other Design Decisions

Maps APIs

Since the app needs to track the position of our users during the day we need to render maps on the operating systems that will support Travlendar+. To be able to do this we will use Google Maps API to track the users position.

Weather Tracking

The application has to keep track of the weather conditions to decide which are the suggested vehicles, which means that we will have to connect the application to an external weather system in order to give the correct information to the user and the application.

3 Algorithm Design

The main task of Travlendar+ is to notify the user of any problem or reminder in his daily schedule. The computation of distances and paths is committed to an external navigation system, which returns only the results. In the following Java-like pseudocode, this external system is referenced as navigationService.

3.1 Notifications

The server keeps a list of candidate notifications with a future timestamp. Every minute, the server looks for the notifications with a timestamp equal to the current time and checks other variables to determine if the notification must be delivered (e.g.: if an event occurs within the same time it takes to move to its location and the user has not left yet, the conditions are met to send a late notification).

```
public class NotificationThread extends Thread {
      while (serverAlive) {
            notifications.check(currentTime);
            wait(60000);
      }
}
```

3.2 External Navigation System

The server interacts with an external actor who provides paths computation. This actor is referenced by the server as an object.

```
public class NavigationService {
     Server server;

     public Path findPath(Location l, Time t, Settings s) {
           Path[] ways = server.ask(l, t);
           // Exclude paths which break settings' constraints
           // Select the shortest path remaining
           return chosenPath;
     }

     [...]
}
```

The objects method findPath returns a Path.

```
public class Path {
      Location startingLocation;
      Location arrivingLocation;
      int distance;
      Time duration;
      Trasport transport;
      set<Path> subPaths;

      [...]
}
```

3.3 Event Creation

Creating an event involves interacting with the external navigation system and adding a new notification to the system.

```
public class User {
      [...]
      Settings settings;

      public void createEvent(Event e, Location loc, Time t) {
            this.schedule.add(e);
            e.editEvent(l, t);
      }
}
```

```
public class Event {
      [...]
      User u;

      public void editEvent(Location loc, Time t) {
            Settings s = u.settings;
            Path path = navigationService.findPath(loc, t, s);
            this.setPath(path);
            Time leavingTime = Time.difference(t, path.duration);
            system.notifications.add(notif.LATE_NOTIF, leavingTime,
path.startingLocation);
      }

      [...]
}
```

3.4 User Settings

The system keeps reference to the users settings in the users object. The settings contain a set of contraints and a method to check them all with given transport and time.

```
public class Settings {
     set<Constraint> constraints;

     public Boolean acceptableTransport(Transport t, Time time) {
          // If there is no contraint on t at the given time, return
true, otherwise return false
     }
}
```

The several types of constraint (lunch, time and transport, transport) extend an abstrat class overriding the public method check.

```
abstrat class Constraint {
      public Boolean check(Trasport t, Time time) {
      // check the constraint with the given data
      }
}
```

4 User Interface Design

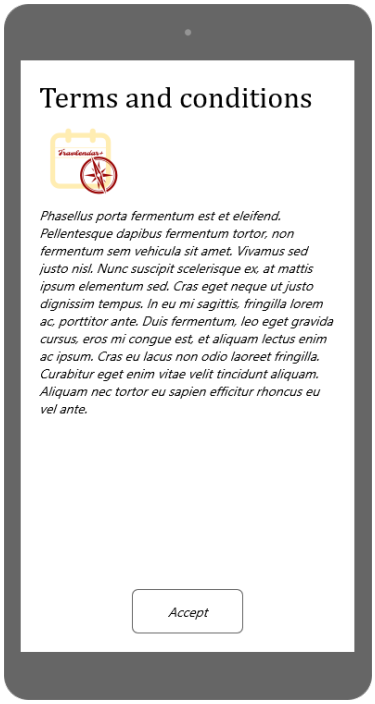
Travlendar+ is mainly designed to be used through mobile application, to encourage its usage in movement. Nevertheless, it is also accessible through web application. There is no exclusive operation which can be done only on one of the two appliances.

4.1 Mockups

4.1.1 Mobile

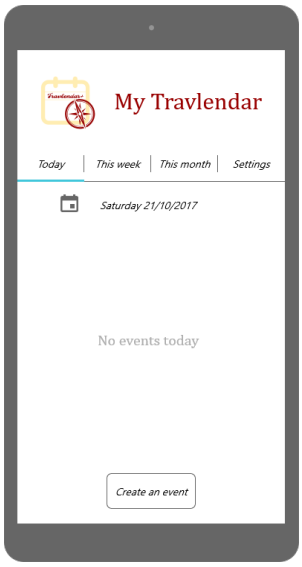
On the first access, it will be requested to the user to login or register to the system. If he is creating a new account, the user must also accept the Travlendar+ terms and conditions.

Mobile app: Account Creation



Mobile app: Terms and Conditions

Once the user logins, he accesses the home page and can create a new event.



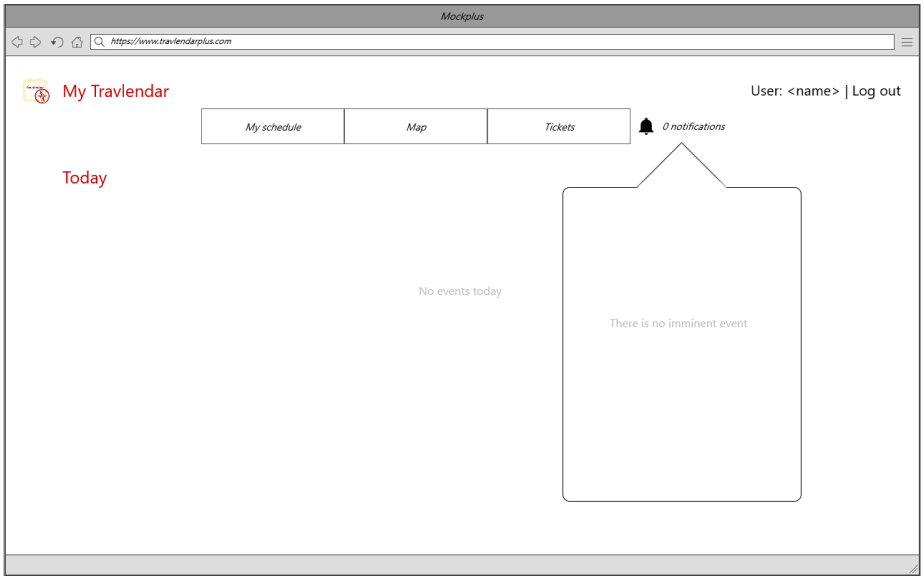
Mobile app: Home Page

Mobile app: Event Creation

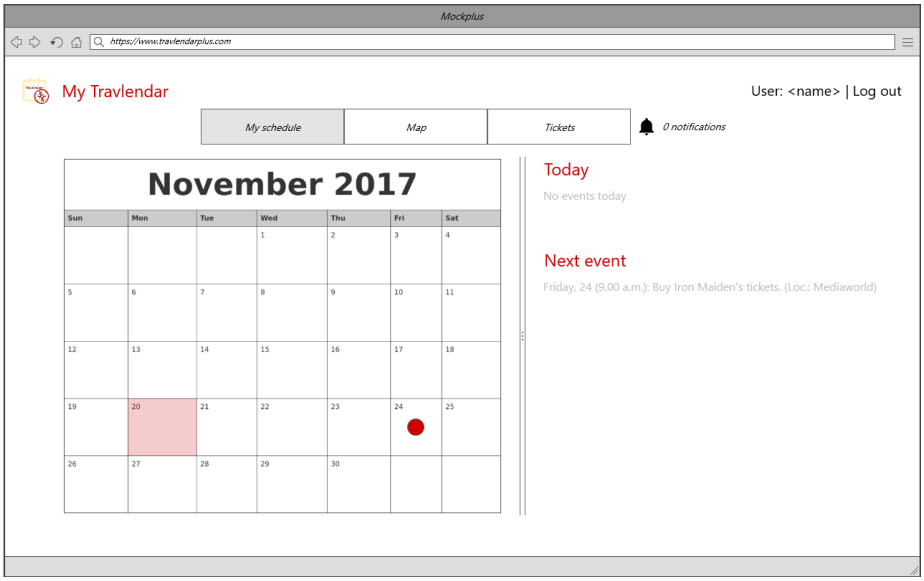
Mobile app: Event Summary

4.1.2 Web

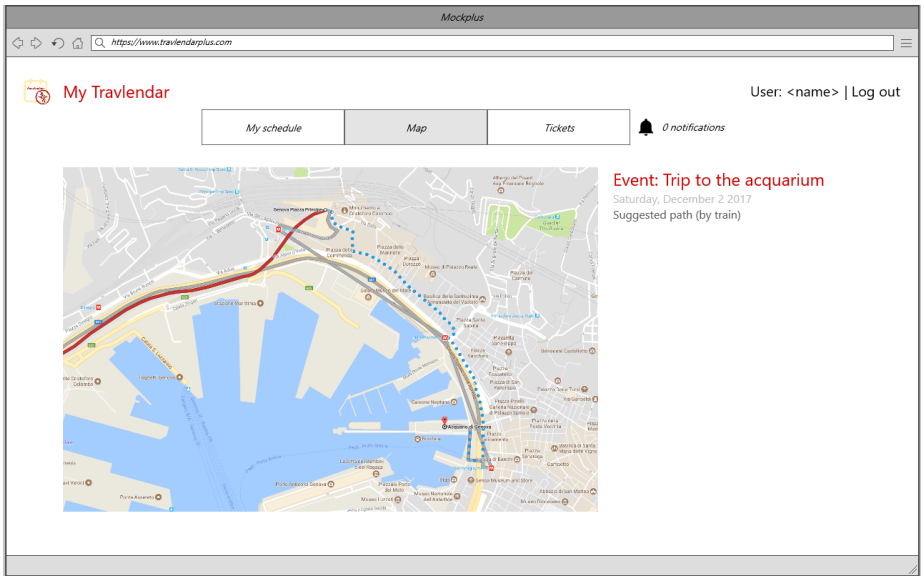
The following mockups will show the user interface when using Travlendar+ from a web browser.



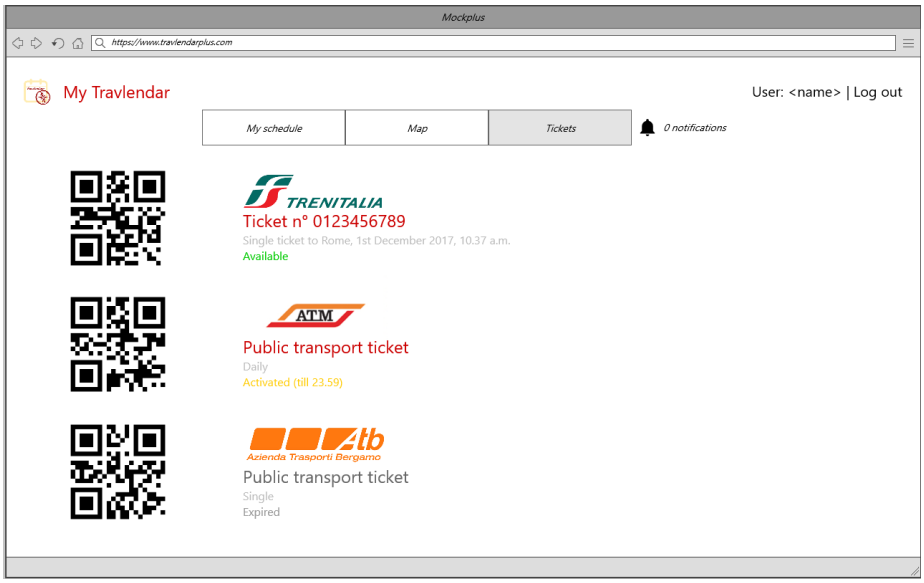
Web: Home Page and Notifications



Web: Schedule



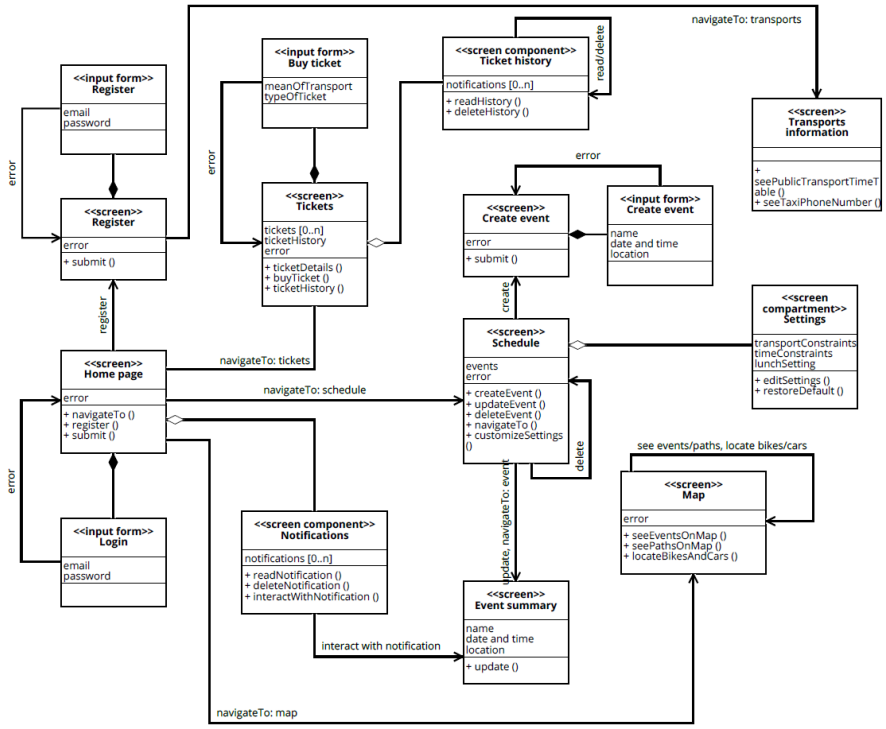
Web: Map showing a suggested path



Web: Tickets

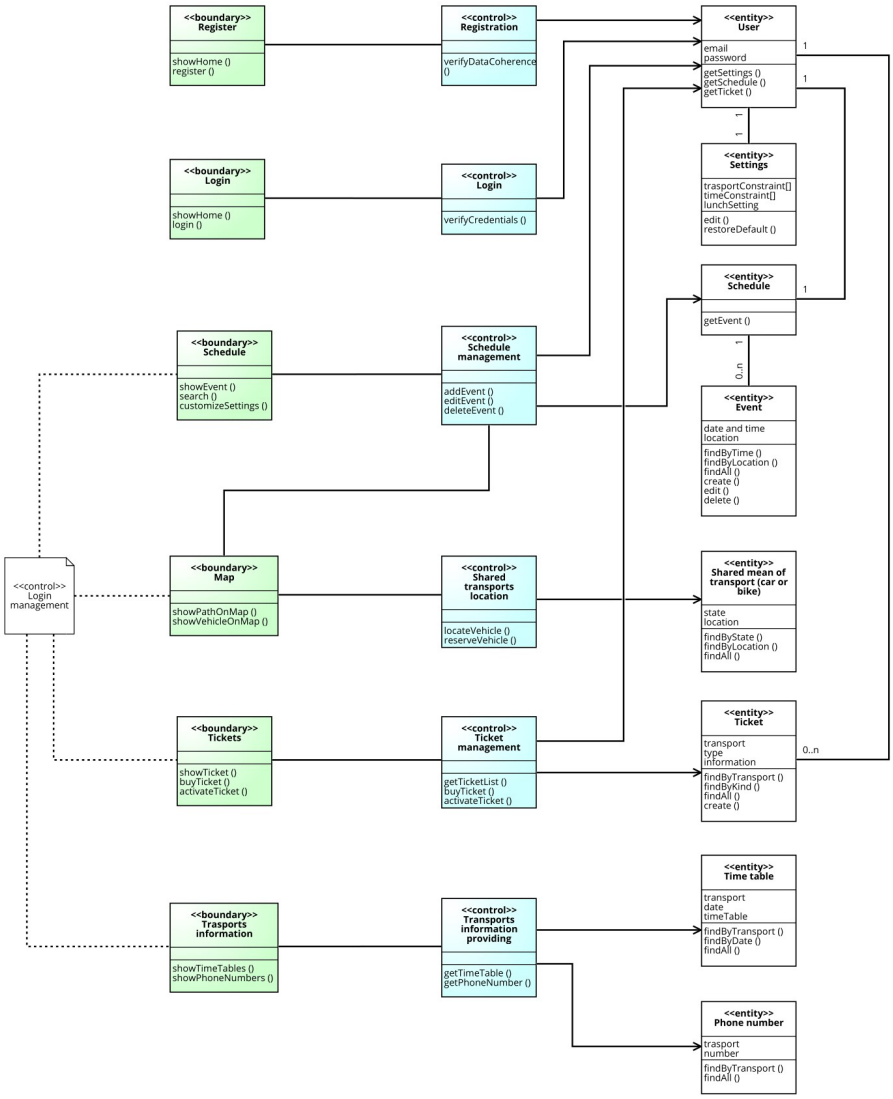
4.2 UX Diagram

The UX diagram provides deep informations about the interaction both with the mobile app and the web app. It is based on the class diagram graphical principles and features 3 kinds of objects: <<screen>> for main pages, <<screen compartment>> for pages elements, and <<input form>> for pages with the purpose to enter users data.



4.3 BCE Diagram

The BCE diagram coniugate the UX diagram with the MVC pattern. The elements of the UX, which belong to the View are linked to the corresponding Control objects which operate on the Entities (the Model, according to MVC).



5 Requirements Traceability

In this section of the Design Document we will analyze the relationship between the components defined above in the document and the goals and requirements that were described in the RASD.

Goal G1: Allow a guest user to register to Travlendar+ by filling the registration form with the data needed.

- User Application Service: Account Manager.
- Requirements from R1 to R4.

Goal G2: Allow the user to select preferences and modify them whenever he wants.

- User Application Service: Account Manager.
- Requirements from R5 to R7.

Goal G3: Allow the user to easily create an organized and customizable agenda based on his preferences.

- User Application Service: Account Manager, Event Module, Schedule Module.
- Requirements R8 and R9.

Goal G4: To help the user plan his movements in a clever and efficient way.

- User Application Service: Account Manager, Schedule Module, Ticket Module, Vehicle Module.
- Requirements from R10 to R13.

Goal G5: To guarantee the user no to be late for his appointments.

- User Application Service: Schedule Module, Ticket Module, Vehicle Module.
- Requirements R14 and R15.

Goal G6: To let the user buy bus or train tickets (both single and seasonal).

- User Application Service: Ticket Module.
- Requirements from R16 to R20.

Goal G7: To let the user find vehicles from vehicle sharing systems.

- User Application Service: Vehicle Module.
- Requirement R21.

Goal G8: Allow the user to buy in advance tickets which can be used later on.

- User Application: Account Manager, Ticket Module.
- Requirements from R22 to R25.

6 Implementation, Integration and Test Plan

6.1 Requirements and Entries

External APIs should be available to guarantee a correct interaction with the navigation service, transport services, ticket dealers and payment gateway.
Most of the components rely on DBMS. This means that the DBSM should be integrated and tested first.
Some components relying on DBMS refer to a specific user, so the Account Manager should be integrated before them.
The Schedule Manager should be integrated after the Event Manager, to be able to test it with concrete data.

6.2 Elements to be integrated and dependencies

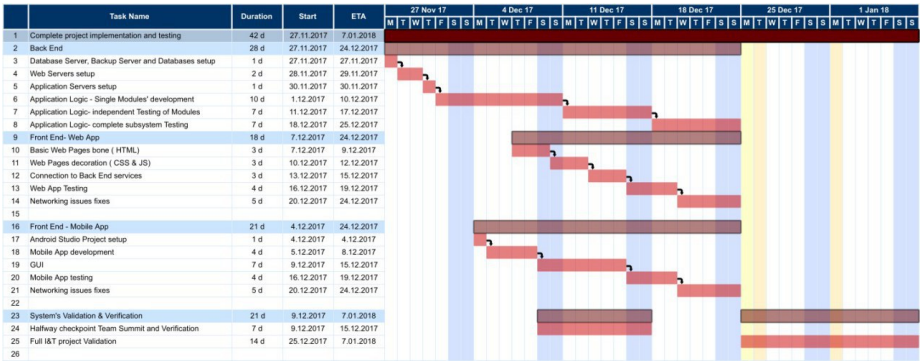
The software components are all enclosed in the User Application Service, the only subsystem. Most of them are independent one from each other, except for the Account Manager which is required by many other components and the Event Manager required by the Schedule Manager.
The components and their dependencies are:

- Account Manager
  - DBMS
- Event Module
  - DBMS
  - Google Maps API
  - Account Manager
- Ticket Module
  - DBMS
  - Payment Gateway
  - Public Transportation Services API
  - Account Manager
- Schedule
  - DBMS
  - Google Maps API
  - Account Manager
  - Schedule Module

- Vehicle
  - Google Maps API
  - Vehicle Services API

6.3 Integration Strategy

Having verified that the external APIs and the DBMS are working, the first components to be integrated should be the Account Manager and the Event Manager, to allow other components to work properly. The remaining components can be integrated in any order, allowing their development to be parallelized. The most suitable testing to start with is bottom-up, so it is possible to check units and subcomponents up to the main component, User Application Service. Critical modules (we consider all subcomponents described in section Components View) integration testing will follow. Server-side of the application has highest priority in our development schedule (see Gantt Diagram provided below in this section).



	Task Name	Duration	Start	ETA
1	Complete project implementation and testing	42 d	27.11.2017	7.01.2018
2	Back End	28 d	27.11.2017	24.12.2017
3	Database Server, Backup Server and Databases setup	1 d	27.11.2017	27.11.2017
4	Web Servers setup	2 d	28.11.2017	29.11.2017
5	Application Servers setup	1 d	30.11.2017	30.11.2017
6	Application Logic - Single Modules' development	10 d	1.12.2017	10.12.2017
7	Application Logic- independent Testing of Modules	7 d	11.12.2017	17.12.2017
8	Application Logic- complete subsystem Testing	7 d	18.12.2017	25.12.2017
9	Front End- Web App	18 d	7.12.2017	24.12.2017
10	Basic Web Pages bone (HTML)	3 d	7.12.2017	9.12.2017
11	Web Pages decoration (CSS & JS)	3 d	10.12.2017	12.12.2017
12	Connection to Back End services	3 d	13.12.2017	15.12.2017
13	Web App Testing	4 d	16.12.2017	19.12.2017
14	Networking issues fixes	5 d	20.12.2017	24.12.2017
15				
16	Front End - Mobile App	21 d	4.12.2017	24.12.2017
17	Android Studio Project setup	1 d	4.12.2017	4.12.2017
18	Mobile App development	4 d	5.12.2017	8.12.2017
19	GUI	7 d	9.12.2017	15.12.2017
20	Mobile App testing	4 d	16.12.2017	19.12.2017
21	Networking issues fixes	5 d	20.12.2017	24.12.2017
22				
23	System's Validation & Verification	21 d	9.12.2017	7.01.2018
24	Halfway checkpoint Team Summit and Verification	7 d	9.12.2017	15.12.2017
25	Full I&T project Validation	14 d	25.12.2017	7.01.2018
26				

Gantt Diagram

6.4 Tools And Test Equiment

6.4.1 Used Tools for Testing

- Junit
- Mockito
- Arquillian
- JMeter

6.4.2 Equipment

Front-end

- **Mobile Application:** the mobile app will be developed for the mobile OS Android using the IDE Android Studio and its integrated simulator.
- **Web Application:** the web application will be tested on several browsers, OS and resolutions.

Back-end

- **Application Server:** the server application will be developed using Java EE with the IDE NetBeans.
- **Web Server:** it will be used an Apache HTTP server.

7 Effort Spent

- Fumagalli Paolo: $\sim$ 30 hours.
- Grotti Pietro: $\sim$ 30 hours.
- Gullo Marco: $\sim$ 30 hours.

8 References

- Google Doc
- Signavio
- Alloy Analyzer 4.2
- StarUML 5.3
- TeXworks