Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Progetto di Ingegneria del Software 2

Parte II: DD

Prof.ssa Di Nitto
A.A. 2017/18

Autori	Matr.
Amadelli Federico	899367
Artoni Alessandro	899343
Baccelli Alessio	898514

Design Document

Document

Progetto *Travlendar+*

1. Introduction

In the Design Document, we give the documentation that will be used to provide software development informations on how the software itself should be built. Inside, there are narrative and graphical documentation of the software design for the project. This includes use case models, sequence diagrams, user's interfaces, object behavior models, and other supporting requirement information.

1.1. Purpose

The purpose of this software design document is to provide a low-level description of the Travlendar+ system, providing insight into the structure and design of each component. Topics covered include the following:

- Class hierarchies and interactions
- Data flow and design
- Algorithmic models
- Design constraints and restrictions
- User interface design

1.2. Scope

This document describes the implementation details of Travlendar+.
The main goal of Travlendar+ is to organize a user's appointments and to ensure that he can participates at every scheduled appointment. In this document we will focus mainly on the component part of our system, the operation of our runtime system and the algorithms used. More information can be found in the RASD , or in the section [5].

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definition

- *Vehicles.* All the means of transport (bike, bus, car...) that the user can use to reach the point of interest.

- *Appointment.* An arrangement to meet someone at a particular time and place, it comprends working appointments, dates, all the events planned by the user.

- *Schedules.* A plan for carrying out a lists of intended events and times.

- *Travel path.* Is the roadway chosen by the system based on the point the user has to go.

- *Accident.* All the event that can happen to delay the user during the travel path.

- *User filters.* The user's preferences on the travel path, like elements to avoid, or elements to take.

- *Id*: identification of the user to access the application

- *Password*: a secret string used by the user to verify his own account

- *Urgency level:* for each appointment is possible to set up a different level, called urgency level, it gives the appointment a certain level of importance, higher is the lever, more important is the appointment, lower is the delay margin.

- *Actor*: an entity that interacts with the system

- *Break:* all types of breaks the user might select to have during the day (e.g. lunch). All breaks are intended to be "flexible": the user has the opportunity to choose a time slot and the duration of the break. The app will then schedule the user's appointments during the day in order to reserve the amount of time specified for the break in the indicated time slot.

- *User:* in our case, when talking about users we're just considering the costumer of our application. This means it is not a two-sided platform application.

1.3.2. Acronyms

- API: Application Programming Interface.

- REST: Representation State Transfer.

- DBMS: Database Management System.

- UI: User Interface.

- RASD: Requirement Analysis and Specification Document.

- ETA: Estimated Time of Arrival.

- MTBF: Mean time between failures.

- MTTR: Mean time to repair.

- UL: urgency level. Is the importance level that the user set to the appointment.

1.3.3. Abbreviations

- DD for Design Document.

- WebApp as for Web Application.

- App for Application

1.4. Revision history

Version	Date	Authors	Summary
1.0	26/11/2017	Amadelli & Artoni & Baccelli	Release
1.1	29/12/2017	Amadelli & Artoni & Baccelli	Added ER Structural minor changes

1.5. Reference Documents

Design Document samples from the beep webpage course or from past year students github.
Slides presented in class on the DD.

1.6. Document Structure

The Design Document is divided into 8 sections with various subsections. The sections of the Software Design Document are:

Introduction	this section introduces the Design Document, it defines the main use of the document and which parts are covered
Architectural Design	this section describes the architectures of our system, how the various components interact between each other and also focuses on individual components to describe their utility
Algorithm Design	this section describes the major algorithms used in the system
User Interface Design	this section describes how the user interfaces of our system

	will look like
Requirements Traceability	this section describes how the requirements defined in the RASD map to the design elements that you have defined in this document
Implementation, Integration, and Test Plan	this section describes the order in which our group plan to implement the subcomponents of our system and the order in which we plan to integrate such subcomponents and tests the integration
Effort Spent	this section describes the hours that each group member has spent in implementing the design document
References	this section contains a list of references our group has taken to create this document

2. Architectural design

In this chapter we will analyze the components that build up Travlendar+ system.
We firstly give an high-level analysis of the structure, showing how the main blocks interact.
Then the single components are explained in details in section [2.3].
In section [2.4] we describe the relation between the logical layers and the physical tiers.
A run-time view of our system is in [2.5] and in [2.6] a schema of the main component interfaces of our system is given.
In section [2.7] we show the main pattern that our system uses, and in [2.8] we discuss some design decision we took.

2.1 Overview

Travlendar+ will be a client-server software, a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.
A 3-tiered structure will be implemented:

- **Presentation Tier** - which is where the user sees information and interacts with the system, the main function of the interface is to translate tasks and results to something the user can understand.
- **Application Tier** - this layer coordinates the application, processes command, makes logical decisions and evaluations and performs calculations, it also moves and processes data between the two layers.
- **Data Tier** - in this layer informations are stored and retrieved from a MySQL database (Accounts, Calendar informations).

We chose this type architecture so we can modify and update the different layers independently.
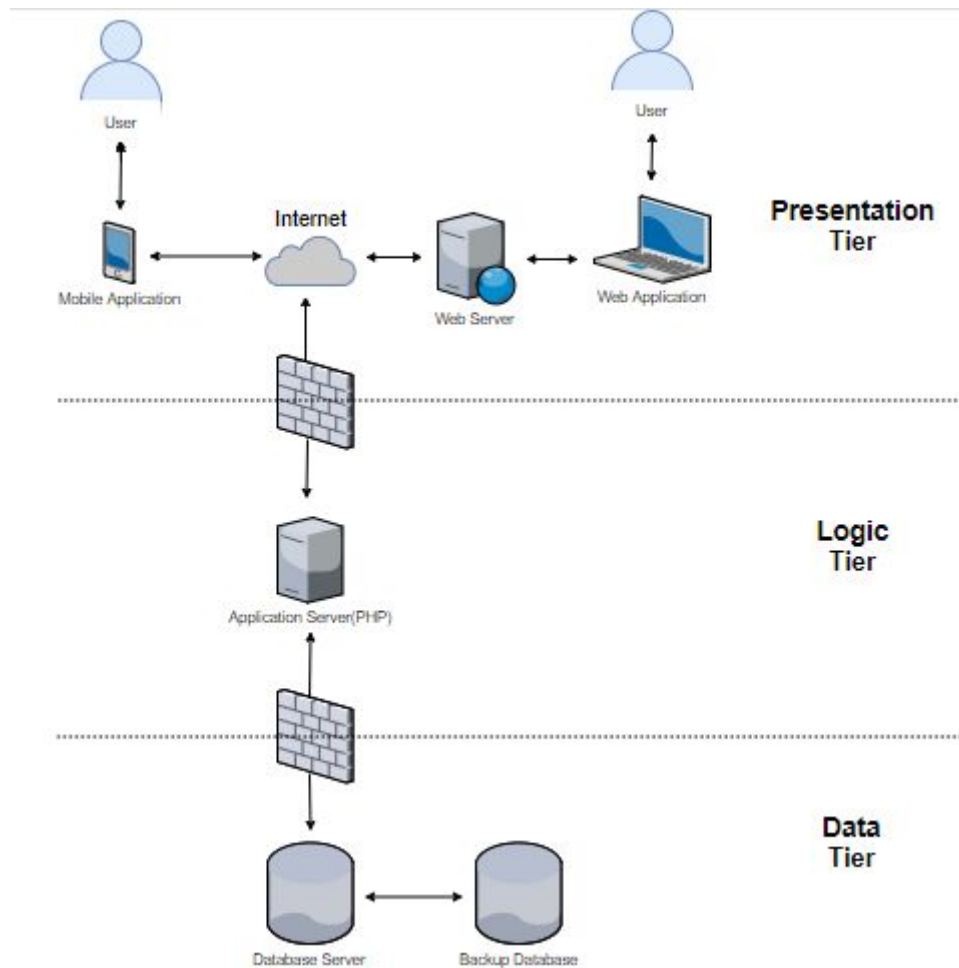
2.2. High level Components and interactions
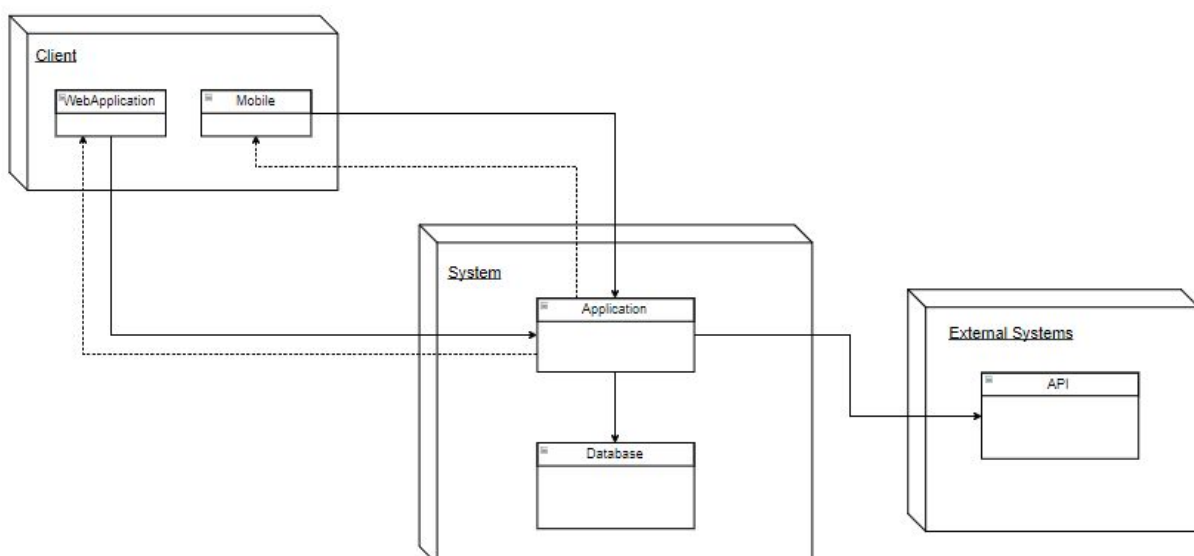
The high level components architecture is composed of 3 different elements types:

- Client;
- System;
- External System.

The main component is the System, it manages the main features of Travlendar+.
The Client delivers to the System the different requests using a mobile app or a web application, the request is synchronous, because the client has to wait the system to acknowledge the request and wait the response of the system.
The System can deliver asynchronous notification to the client to inform him of some accidents that can modify the schedule of the client, the notification is asynchronous because the system has only to notify the client without waiting for his response.
The System has to request his database to obtain old information for the client
The System has also to request external systems to get information of the external world using API, there are 2 different types of this request:

1. requesting information following a client request, this type of request must be synchronous as the system can not respond to the client until it has the information;
2. to request information about the outside world periodically, to be up to date on possible accidents that may interfere with the client's path, this type of request may be asynchronous because the information is not needed for the work being done but is just a request routine and if you have a positive response, that is, accidents on the path, the system will send a notification to the user

2.3 Component view

Below is how our system is organized and what components are used.
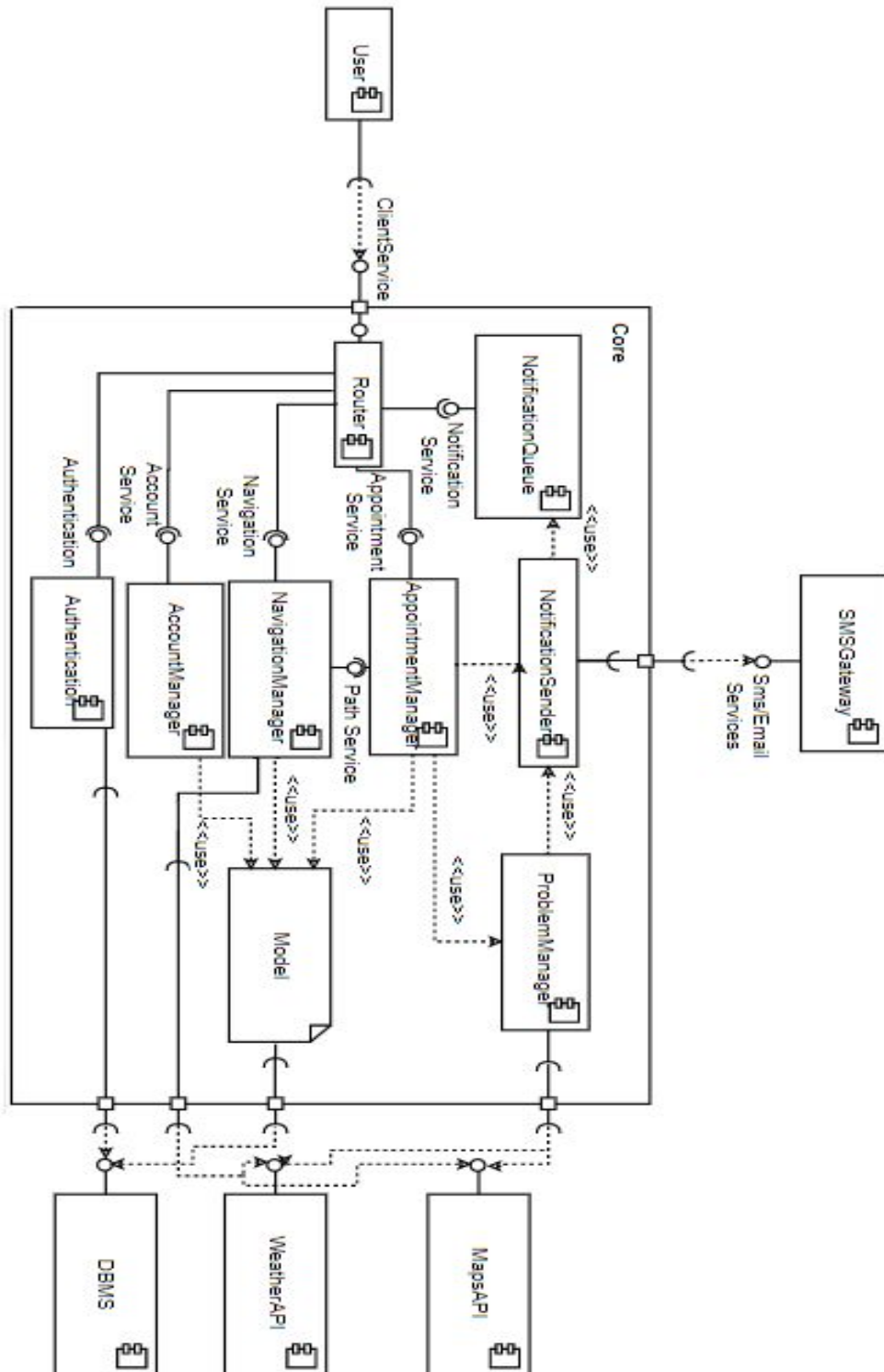
The 8 main components of the core are:
- **Router**: It is an internal network node, it routes the various customer requests to the component that meets the required service, always receives HTTP requests and analyzes it to understand the user's demand;
- **Authentication**: component needed to verify whether the data entered by a user corresponds to the data of a registered account in databases;
- **NavigationManager**: creates for the user the path to reach the appointments that the user has scheduled on his calendar, it is connected to MapsAPI, to build the path, and to the model, to know the user's preferences;
- **AppointmentManager**: this component is responsible for managing appointments, manages the addition of new appointments and chooses where to place mobile appointments (lunch, break)
- **NotificationSender**: this component is responsible for sending various notifications that the system has detected to the user through:
  - passive notifications: which the user receives when the app opens;
  - active notifications: which the user receives if he has active internet, or the web page open;
  - external gateways: that inform the user via other means (email, SMS).
- **NotificationQueue**:this component saves the various notifications to show when the user connects.
- **ProblemManager**: this component is responsible for detecting errors, detecting new user's appointments, and enrolling on the various events that the user will use in his path, if a problem is encountered for the event that the component is logged in, it checks the issues which leverage would cause the user and inform the NotificationSender.
- **AccountManager**: this component is used to modify or add: data, user preferences, vehicle passes ... within the user's account.

the last object that was put into the system is:
- **Model**: it is the vision that the system has of our database and with which data it interacts.

In the next section, we will take a closer look at the most relevant components in our system, namely: AppointmentManager and NavigationManager.
We also put an ER schema of the database handled by the DBMS component.

2.3.1. AppointmentManager
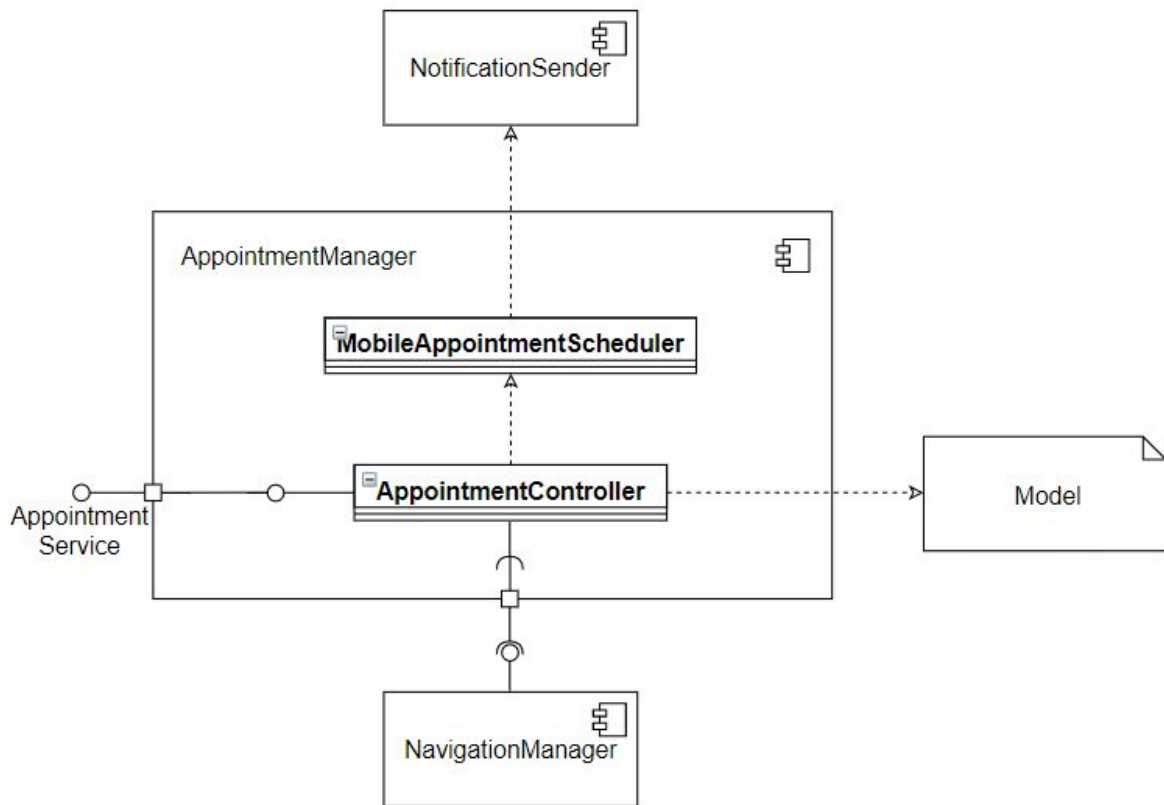
As mentioned in the previous section, the Appointment Manager manages the appointment part, it consists of 2 parts:

- **AppointmentController**: it controls appointments and the various options that a user can make about them(adding, deleting, revising...);
- **MobileAppointmentScheduler**: it takes care of positioning and managing mobile appointments, that is, those appointments that are to be made within a given time interval and they must last for a certain number of minutes. Then the component must check according to the scheduled appointments when the mobile appointment can be placed, and, if it is not possible to allocate it anywhere, it sends a notification to the user.

2.3.2. NavigationManager
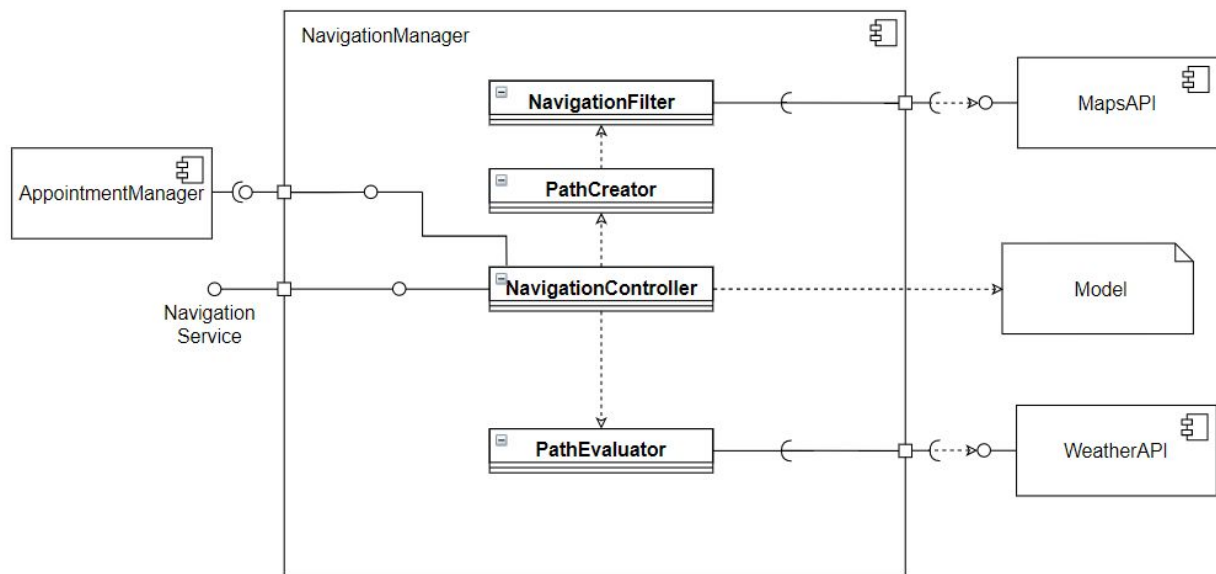
The navigation manager manages all the content of the navigation to arrive at certain appointments.

It consists of 4 parts:

- **NavigationFilter**: it filters all the solutions that contain the means of transport that the user has banned;
- **PathCreator**: it serves to find the best paths that the user can use to go to the next appointment
- **NavigationController**: it monitors the interaction of the various components to provide the path that suits the user more
- **PathEvaluator**: it receives the paths found and evaluates them according to various criteria such as weather, cost, time, user preferences.

2.3.3. DBMS component

The DBMS component manages all the datas that needs to be store in our application.
In order to implement this component, any relational DBMS can be used such as MySQL or PSQL.
Access to data must be secure: only authorized user with correct credential can retrieve their personal calendar.
The schema of the database is represented by the following Entity Relationship diagram.
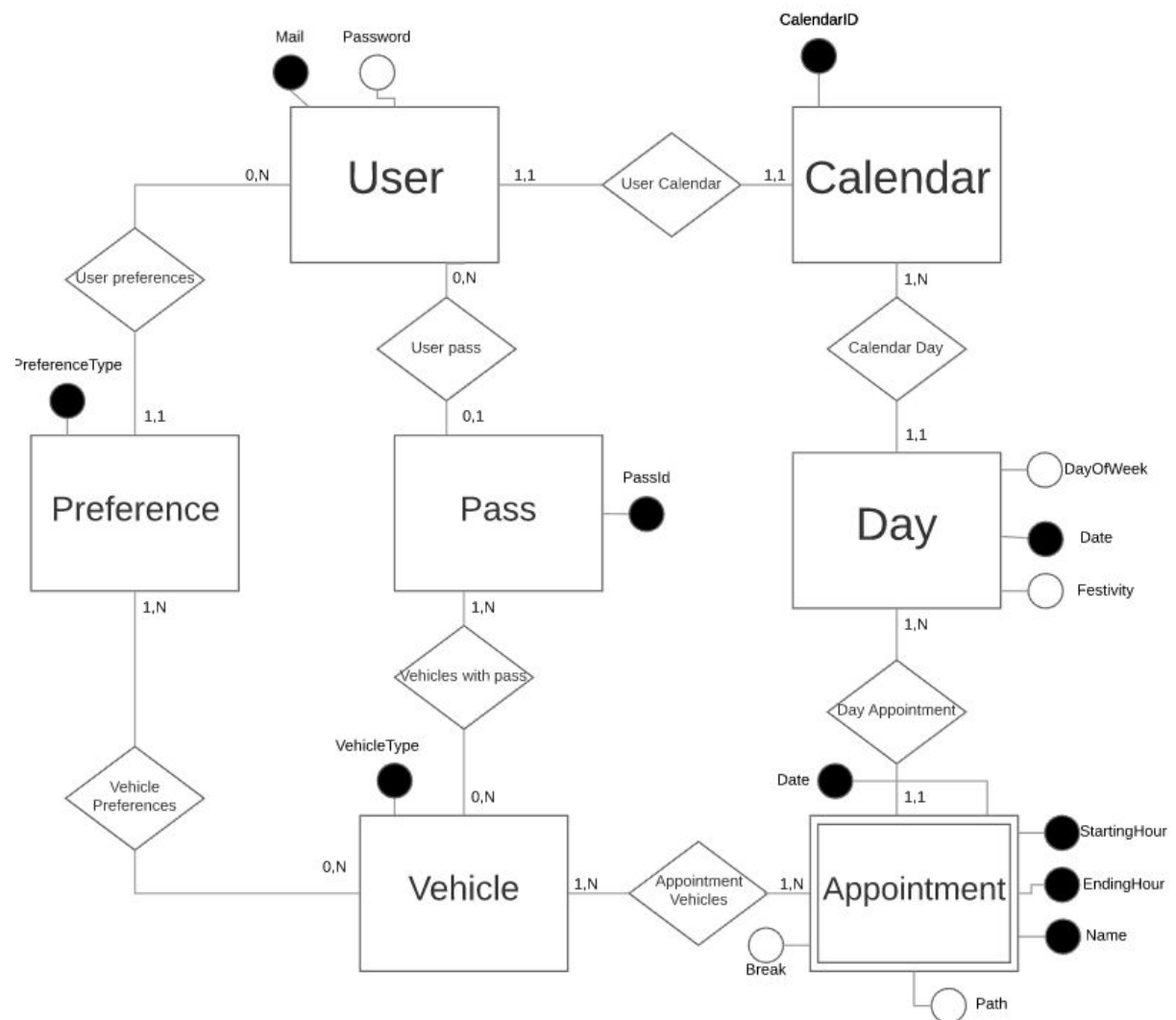
2.4 Deployment view

The deployment view illustrates the distribution of processing across a set of nodes on the system.
The Web Browser and the MobileApp - that represent the Presentation Tier - communicate with the server application - the Logic Tier using REST messages. The application server communicates with the database - the Data Tier - using SQL.
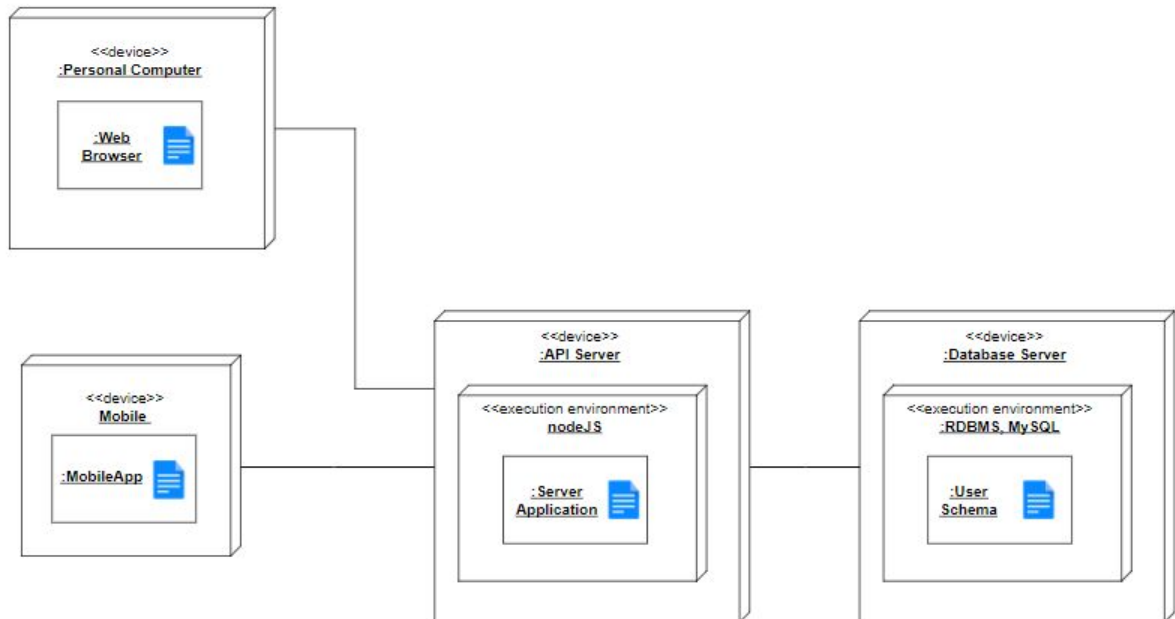
2.5. Runtime view

Below there are several sequence diagram to better understand the interaction of the components and the functions they call when they have to perform a given task.

2.5.1. Add appointment

This is the interaction that has the components between each other to add an appointment, the chart starts already when the target adds information in the specific form to add the appointment, there is therefore no call to add the appointment and the passage of the its module as they are rather trivial.
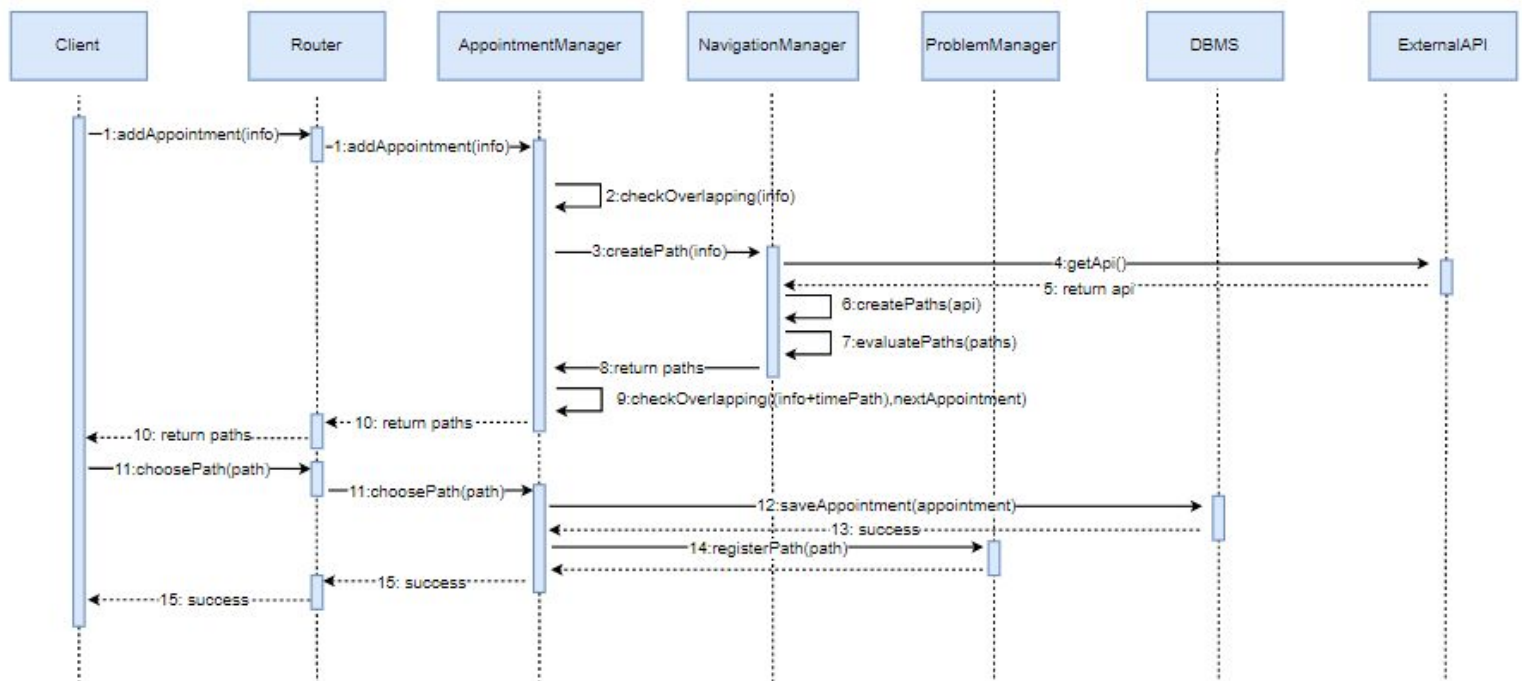
2.5.2 Problem on a registered event

This is the interaction between the components when the *external API* informs the *Problem Manager* that a problem occurred in one of the events to which it was registered, so the event registration (which can be seen in the previous paragraph) and also the notification that is sent to the *Problem Manager* to inform about the problem have already occurred before the interaction
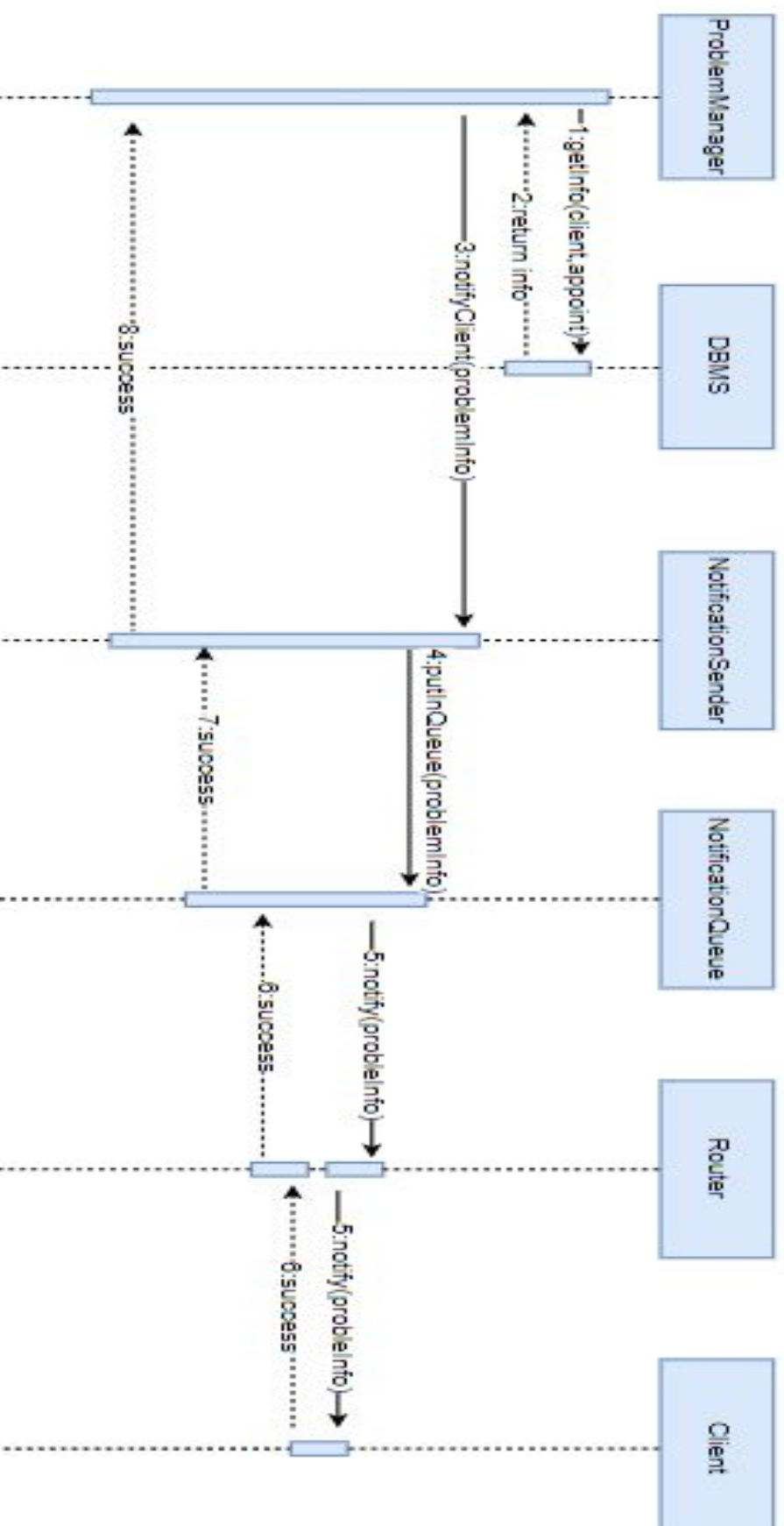
2.6 Component interfaces

2.6.1 Front end to application service

The Web application and the Mobile Application must communicate with the Application server through a RESTful interface. JSON is used as the data representation language. The REST resources and the functionalities offered will be implemented in this way:
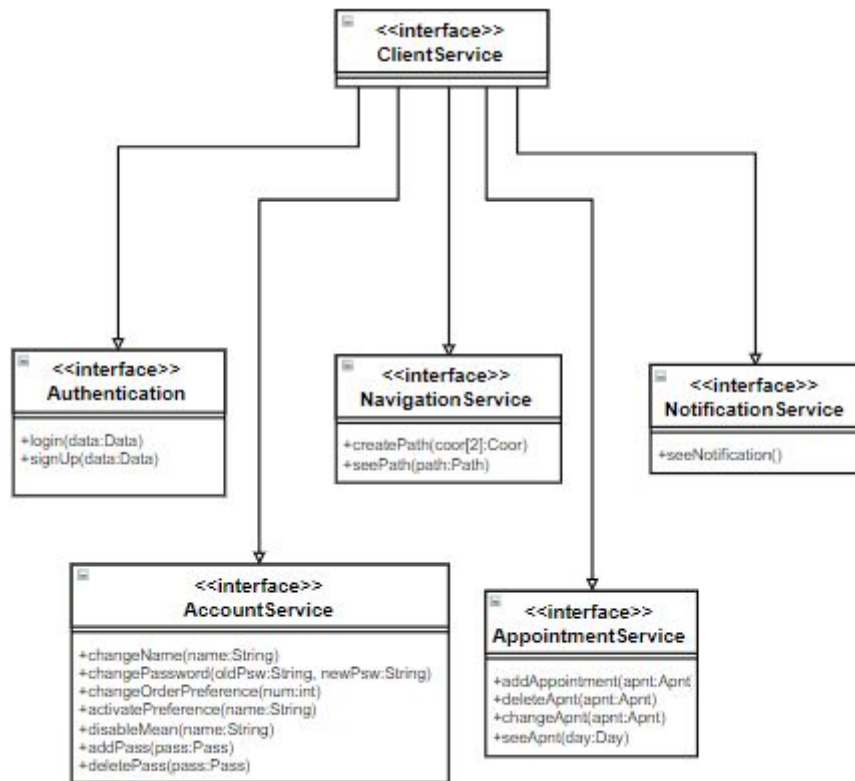
2.7 Selected architectural styles and patterns

The following architectural styles will be used:

- **Client- Server**. The front end of our application are users communicating with the application server. Clients make a request and our server processes it, retrieving informations from the database when needed.
- **Service-Oriented Architecture(SOA).**. Used
- **Model-View-Controller (MVC)**. MVC is followed throughout the whole system design. our Presentation Tier is the View, the Application server is the controller that can retrieve information from the Database (model) when needed.

We also highlighted

- **Adapter pattern.** Users when looking for their best time travel will result in a sort of a query into the server that will interrogate Google's API and answer considering the user vehicle preferences. An *adapter pattern* will be used to fit properly the users questions to what google API can answer.
- **Publish/subscribe pattern.** By adding a certain appointment the system will check if it can be subjected to some problems (such as strike) and in case it adds the account-id to a particular list of persons that need to be notified if that event occurs. So if a strikes occurs in a specific day, and the user planned to use a strike-subjected vehicle, the application will notify the user that will act properly. This whole process will be implemented using a *publish/subscribe pattern.*

2.8 Other design decisions

2.8.1 Password registration

Users' password won't be stored plaintext but they will hashed and then stored in our database.

2.8.2 GPS

The main goal of the application is to help the user managing his schedule and it is not intended as a navigator GPS device. Because of this, the system merely computes and proposes the path to the user but is not following him with the GPS along the path.

2.8.3 Overlapping events

In our view, the user should have the complete control over the application, which is intended to support him in the management of his schedule. Therefore, the user is de-facto allowed to add an event which overlaps with another one or which is not possible to reach on time given the starting position and the travel time. In these cases, the application will send a notification that a problem in the schedule may arise but will let to the user the choice on what to do;

2.8.4 External APIs

We also decided to rely on external APIs, like Google Maps APIs because we found extremely thought to create our own maps system, and because it also provides a very good and precise path/ ETA.

We didn't choose google Calendar APIs in order to help our application development because we found Google Calendar is usually an external component that can be added in some website but it doesn't really give you the freedom to use a proper calendar.

2.8.5 NodeJS

We decided to use NodeJs because we find "Single Threaded Event Loop" more easy to handle multiple concurrent client requests allowing a great scalability.
Also, since it uses less threads, NodeJs applications typically use less memory resulting also more efficient in a client/server application with little to compute.
With NodeJs, we also use sqlite3, express and other dependencies that can be found in package.json.

On top of NodeJs we will use HTML5, CSS3, JQuery for our Web Application.

3. Algorithm design

Different algorithms will be designed and implemented in order to achieve the goals of the system.

- **Path Computation**: one of the main functionalities that is provided to the user is the travel path to go to an appointment. The calculation of the path is done through an external API [RASD, sec 1.5.1], which will be used by Travlendar+ everytime a new calculation is needed. This process will take into account different preferences and settings that the user can set. In all these cases is important to notice that only the paths which are fast enough to reach the destination on time will be proposed. The preferences are:
  - Global preferences: the user has the possibility to set general preferences on vehicles (e.g. deselect car). If the user decides to deselect a particular travel mean, when comparing the different paths the API will return it will automatically exclude the ones with the deselected vehicle;
  - Preferences for the travel: the user can select vehicles we would prefer to use during the travel. These preferences prevail over the global preferences (e.g. a person without the driving license may have deselected the car globally, but for a certain appointment he knows he will be driven by car by a friend of him and therefore selects the car among the preferences for the travel) and are taken into account when the system will have to display to the user the best possible path. The system will try to find a path with the vehicles he selected among the preferences and will then display the quickest one.
  - Weather: the system will also take into account the weather thanks to an external API [RASD, 1.5.3] which will provide the forecasts. Indeed, based on the weather the system will offer paths that it feels more comfortable and convenient for the user, trying to avoid unpleasant situations. For example, if the weather according to the forecasts is rainy during the travel, the system will minimize the time the user has to spend outside walking by foot by comparing the different paths the API will return as result;
  - Lowest Carbon Footprint: the user has also the possibility to ask the system to compute a path which minimize the carbon footprint. First of all, in the application database will be inserted the data about the footprint for the different vehicles [RASD, sec. 1.5.2]. Then, the system will compute the estimated footprint for the various paths the API returns and will suggest to the user the one which has the lower impact on the environment.

Here it is a snippet of code:

```
preferencesList = getUserPreferences();          //specific preferences for that travel
globalPreferences = getGlobalUserPreferences();       //globally deselected vehicles
pathsList = GoogleAPI.computePath(startingLocation, arrivalLocation, startingHour);
lowestCarbonFootprint = getFilter();
if (lowestCarbonFootprint is true)
   pathsListOrdered = sortPathListByCarbonFootprint(pathsList)
else
   pathsListOrdered = sortPathListByTravelTime(pathsList);
bestPathFound = false;
for (elem in pathsListOrdered){
   for (vehicle in elem.vehicleList){
      if (vehicle in globalPreferences and in preferencesList)
        pathsList.delete(elem);
        break;
      if (vehicle is contained in preferencesList)
        bestPath = elem;
        bestPathFound = true;
        break;
   }
   if (bestPathFound is true)
      break;
}
display bestPath;
```

Since we're going to use this "Single Threaded Event Loop" Architecture here it is an
example of a pseudo code event Loop:

```
public class EventLoop {
while(true){
      if(EventQueue receives a JavaScript Function Call){
              ClientRequest request = EventQueue.getClientRequest();
               If(request requires BlokingIO or takes more computation time)
                  Assign request to Thread T1;
               Else
                  Process and Prepare response;
        }
      }
}
```
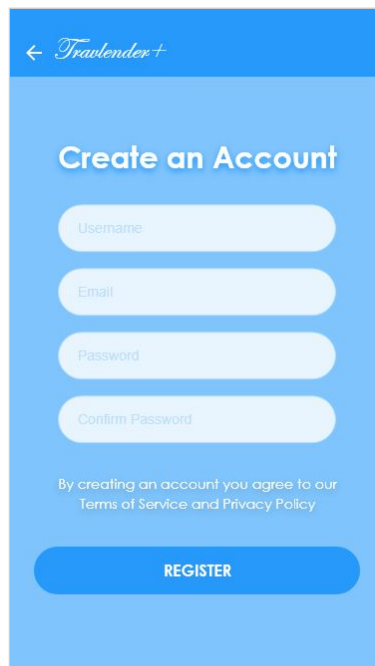
4. User interface design

The interface of the user consists in different menus, thanks to which the user can interact with the system. Below is a list of all the various screens. Those are preliminary mockups of a mobile view web application.
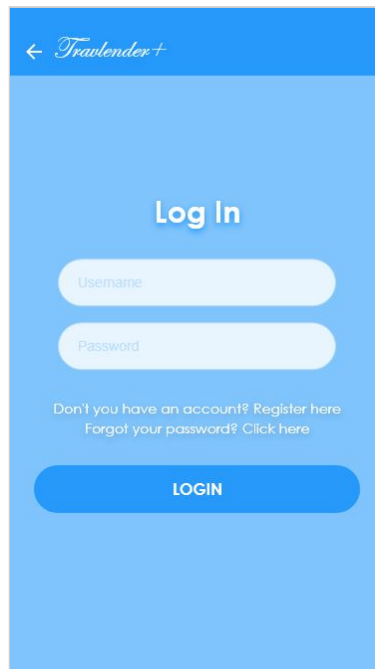
4.1. MockUps

- **REGISTRATION:**

If a user uses the application for the first time, the registration menu appears, asking him all the data which are needed in order to complete the registration, such as username, email and password;

- **LOGIN:**

The login screen will become allow an user who has already done the registration to access his personal profile and calendar. In order to log in has to insert a valid username and password to complete the login successfully.

- **HOME:**

Once the user has logged in, he will be redirected to his personal home page. From here he can see the full schedule of the week and the weather forecasts of his town. By pressing the arrow in the top right corner he can see the schedule of the coming weeks, while by pressing a specific day will allow him to check all the details of his appointments. At the bottom of the screen a "Profile" button is available, to allow the user to change the settings and his information.

-   **DAY SCHEDULE:**
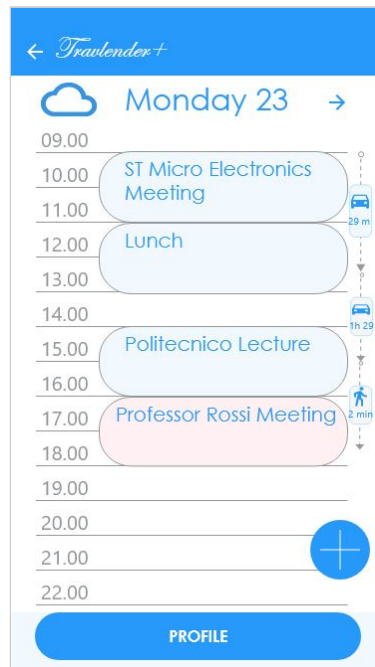
From here the user can see a more detailed schedule of his appointments in a specific day. By clicking on the arrow in the top left part of the screen he can go back to the home, while the arrow in the top right corner redirect him at the next day. Moreover, it is possible for the user to add a new appointment by clicking on the "+" button, while it is still available for him the possibility to change his personal information and settings through the "Profile" button.

-   **ADD EVENT:**

If the user decides to add an appointment, he will be asked to fill a form with the data about the appointment itself. The system requires a name for the event, the time slot, the location of both the starting point and appointment and gives to the user the possibility to choose a preference for a vehicle for the specific travel. Moreover, the

user has the possibility to create a flexible lunch or a flexible break, by selecting one of the two options "Break" and "Lunch" and also to assign a level of importance of the appointment.

- **TRAVEL PATH:**

In this screen the user can see the path he has to do and the vehicles he has to take to go from the starting point to the location of the appointment.

- **PROFILE:**
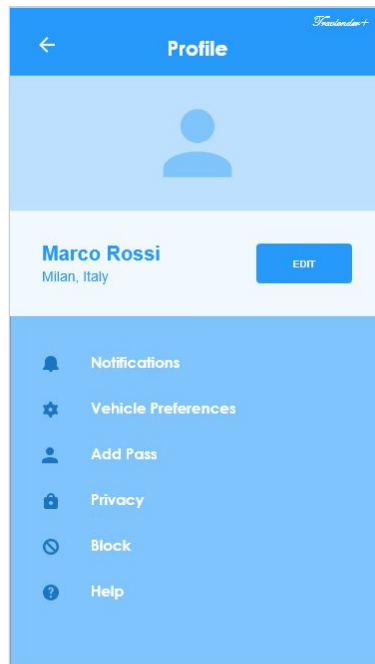
In the "Profile" page the user can change the settings, edit his personal information (change username, password, email, etc) and also change the vehicle preferences and add a pass for the public transportation.

- **VEHICLES PREFERENCES:**

Here the user can change his vehicle preferences. He can select/deselect any travel means and in the future the system will not show the user paths with the deselected vehicles.

4.2. UX
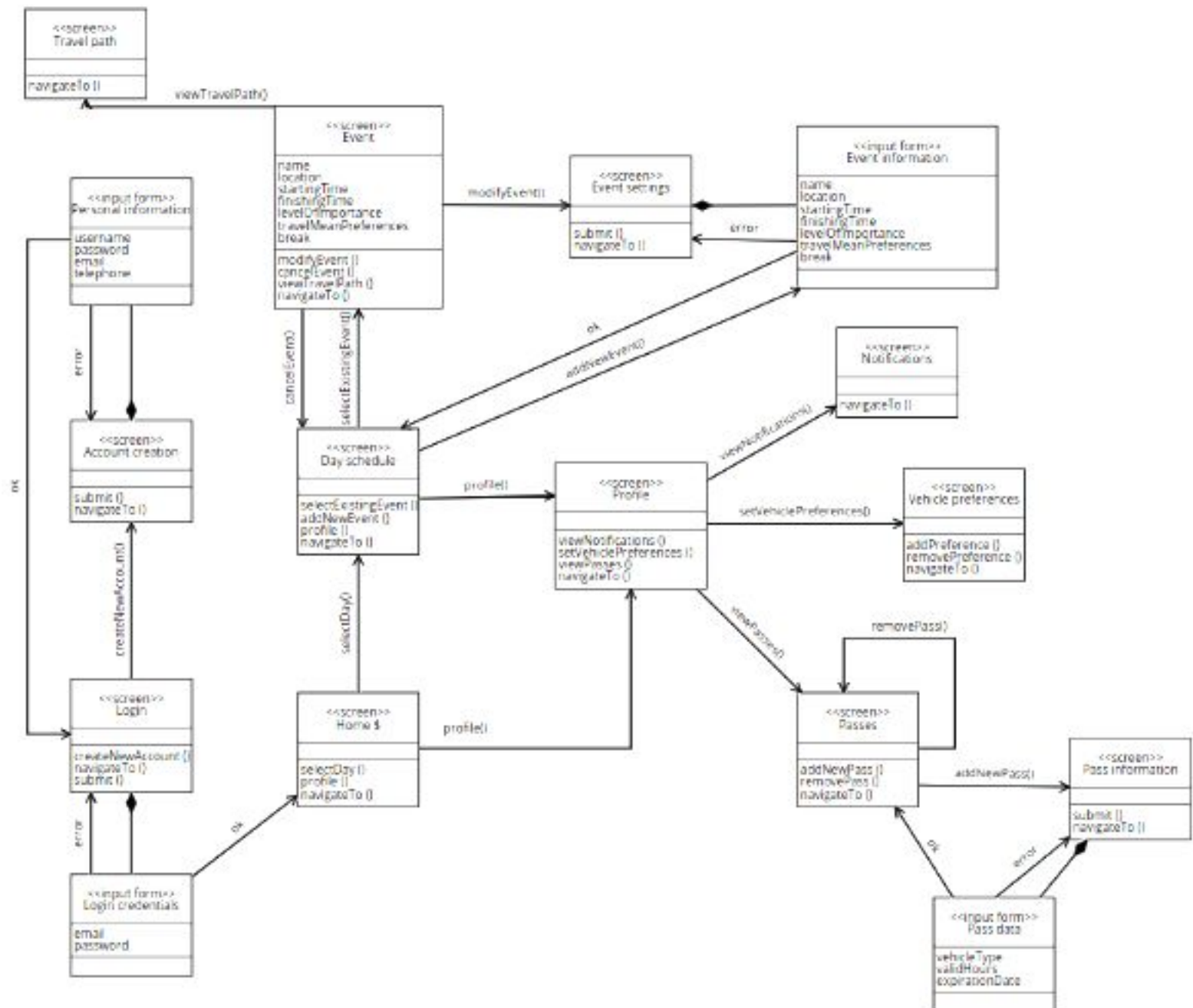
4.3. BCE

4.3.1. Access to The Application

4.3.2. Application

5. Requirement traceability

The main goal of this document, and then the Design choices presented, was to fulfill in a complete and correct way all the goals specified in the RASD.
The following list will provide a mapping between goals and requirements defined in the RASD and system components illustrated in [2.*] section.
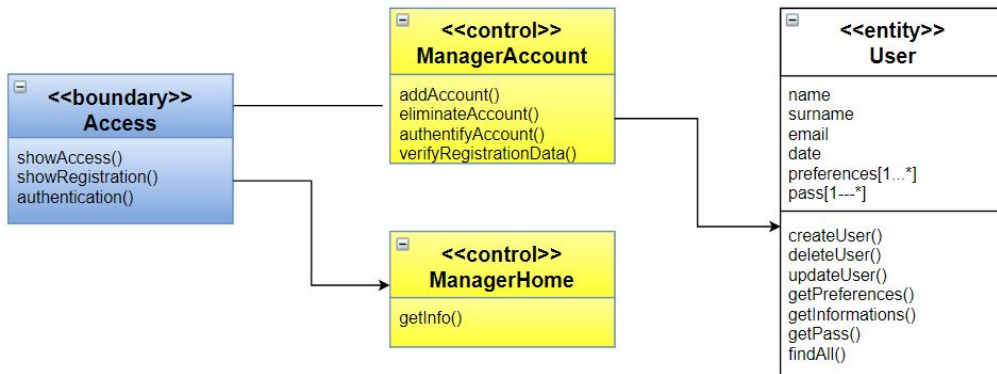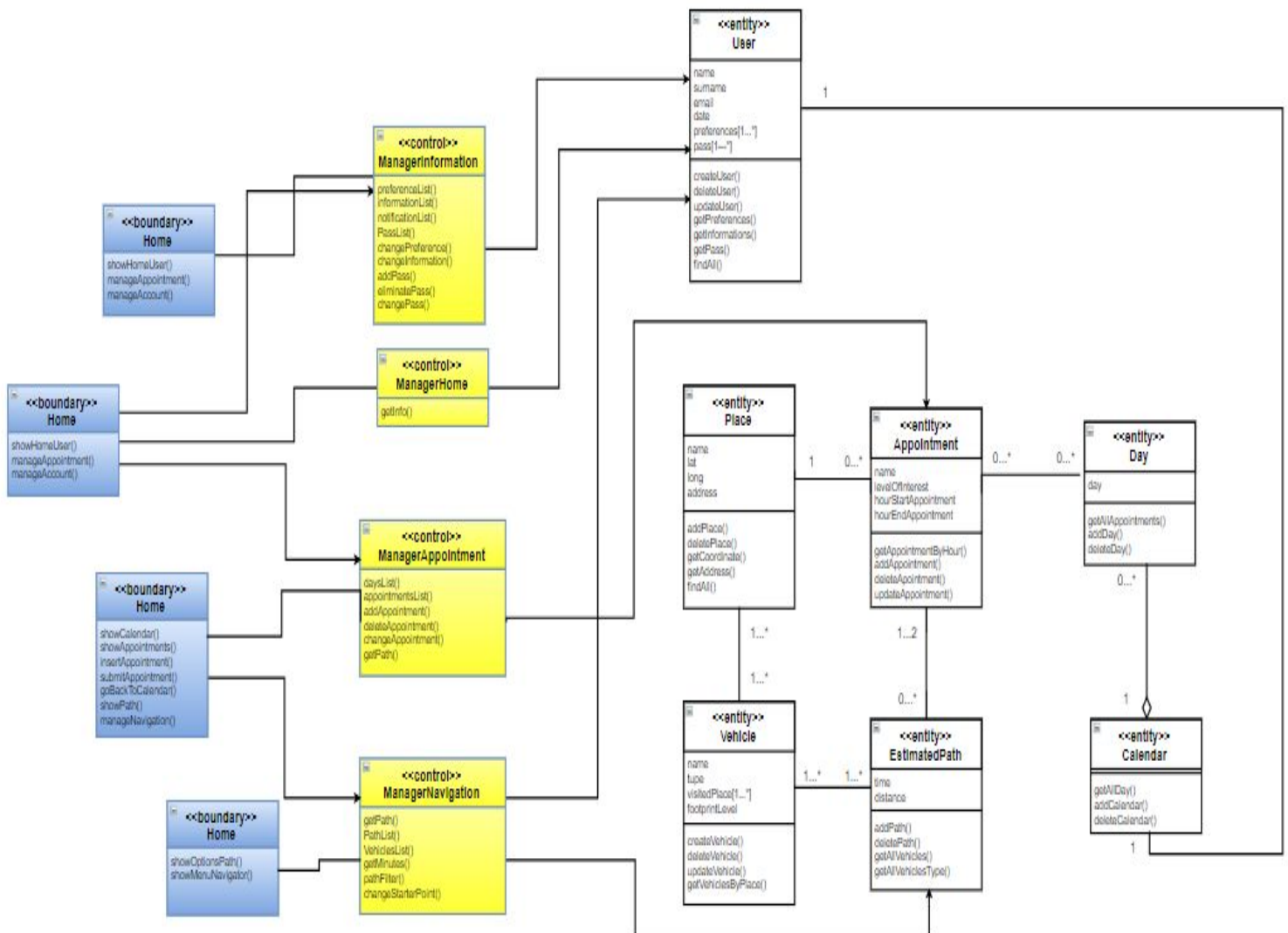
[GOAL0]   Allow users to login and register into the Web Application
    Requirements: [REQ0 - … - REQ3]
    Components: Authentication

[GOAL1.0] Allow users to build a calendar where they can add appointments;
    Requirements: [REQ1.0 - … - REQ1.4]
    Components: AppointmentManager, NavigationManager, AccountManager

[GOAL1.1] Give the possibility to schedule particular appointment
    Requirements: [REQ1.5 - REQ1.6]
    Components: AppointmentManager

[GOAL1.2] Set a "Level of importance" of a certain appointment. The application will act differently from low importance appointment to high importance appointment
    Requirements: [REQ1.7]
    Components: AppointmentManager, ProblemManager

[GOAL2.0] Identify a best path solution in order to reach one appointment from another;
    Requirements: [REQ2.0 - … - REQ2.2]
    Components: NavigationManager, AppointmentManager

[GOAL2.1] Allow clients to filter the travel solution to select the one they prefer;
    Requirements: [REQ2.3]
    Components: NavigationManager

[GOAL2.2] Show all the possible travel solutions to arrive at the next appointment
    Requirements: [REQ2.4 - REQ2.5]
    Components: NavigationManager

[GOAL3.0] Support users in their travel.
    Requirements: [REQ3.0 - REQ3.1]
    Components: AccountManager

[GOAL3.1] Allow user to buy tickets from the Web Application
    Requirements: [REQ3.2 - … - REQ3.3]
    Components: AppointmentManager

[GOAL4.0] Support users in their everyday life trouble, for example notifying a user that when it rains he needs to take an umbrella or in case of strike to consider to take another vehicle.
Requirements: [REQ4.0 - … - REQ4.3]
Components: NotificationSender, ProblemManager

[GOAL4.1] Tell users what the weather is in a certain day
Requirements: [RE4.3]
Components: WeatherAPI

6. Implementation, integration, test plan

We defined some tasks we believed important, and we estimated the time we believe it will take to do that particular task.

Since it is only an estimation, we reserved a lot of time (more than 2 weeks) to review the system and to improve it.

We expect to deliver an alpha version of the system with some functionalities, having a proper Website running.

Tasks	26-nov	30-nov	04-dic	07-dic	11-dic	14-dic	18-dic	21-dic	25-dic	28-dic	01-gen	04-gen	07-gen	11-gen	14-gen
Login & Registration	▓														
Database Creation	▓														
Calendar Creation		▓	▓	▓											
Calendar Logic		▓	▓	▓	▓										
Navigation Integration		▓	▓	▓	▓	▓									
Testing AddEvent		▓	▓	▓	▓	▓									
Flexible Lunch						▓	▓								
Level of importance						▓	▓								
Vehicle preferences						▓	▓	▓							
Weather / Strikes															
Ticketing System							▓	▓	▓						
Testing AddEvent2									▓	▓	▓				
Implementing notification/Subscribe								▓	▓	▓					
Checking IDT specs/ Improving Software									▓	▓	▓	▓			
Creating ADT											▓	▓	▓	▓	▓

We identified the "Navigation" and "Appointment" components as the most significant ones so the "Calendar Logic" and the "Navigation Integration" processes will be one of the most significant ones, and nearly the first that will be implemented.

If we spend too much time implementing these components, we will consider the project failed and therefore we will abandon it.

Also, the testing of that task will cover most time of the project.

Another task we believe it's important, it's the notification system. Still, we think that in order to make notifications, first it is important to have at least a working and well structured system, so we decided to put the implementation of this task as one of the lasts.

One of the key part of the whole project will be the integration of the system with the API's we decided to use. The most important one, the Google Maps API, will be integrated first; then we will focus on the integration of the weather API, which we identified to be less important of the former API.

Test planning will then cover mostly those two tasks. Still little snippets of code to test if all components are available (for example if database is connected ..) will be putted in order to guarantee a good test-coverage.

7. Effort spent

Students	Hours spent
Federico Amadelli	18.5 hrs
Alessandro Artoni	12,0 hrs
Alessio Baccelli	9,5 hrs

8. References

F.Amadelli A.Artoni A.Baccelli: RASD, Software Engineering Project Part I, 2017
E. Di Nitto: Mandatory Project goal, schedule, and rules