POLITECNICO
MILANO 1863

Travlendar+ project

Design Document

RICCARDO FACCHINI

ANDREA GUGLIELMETTI

November 11, 2017

Deliverable specific information

Deliverable:	Design Document
Title:	Requirement Analysis and Verification Document
Authors:	Riccardo Facchini - Andrea Guglielmetti
Version:	1.0
Date:	November 11, 2017
Download page:	https://github.com/Riccardo95Facchini/FacchiniGuglielmetti.git
Copyright:	Copyright © 2017, Riccardo Facchini - Andrea Guglielmetti – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document aims to detail the design of the software and of the architecture regarding the system of Travlendar+. To do so it will be taken a more detailed approach for the description of each component and the overall architecture of the system, by also pointing out the relations between each module and giving a description of such relationships.

1.2 Scope

Travlendar+ is a time/travel management web-based system, designed to help the users to keep track of their daily routine by scheduling for them the best way to move from one place to the other using all the information given by the users themselves and external data in order to deliver a tailored experience for everyone.
After the user enters all the needed data to register an event, the system will automatically alert him/her when it's time to leave and will give directions to reach each one of the means of travel as specified in the options, taking into account also factors like the weather and possible public transportation strikes.

1.3 Definitions, Acronyms, Abbreviation

- DD : Design Document
- RASD: Requirement Analysis and Specification Document
- API: Application Programming Interface
- DB: DataBase
- DBMS: DataBase Management System
- GPS: Global Position System

1.4 Revision History

1.5 Reference Documents

- Document of the assignment: Mandatory Project Assignments.pdf
- Requirements and Specification Document

1.6 Document Structure

2 Architectural Design

2.1 Overview

We need to design a system in which the user asks to the system to store an appointment and calculate the best path from a starting location to the appointment location.
Since this interaction between user and system can be summarize as:

1. User request a service to the system.
2. System responds to the user with the requested service.

Based on this, we decide to use a client-server architectural approach.

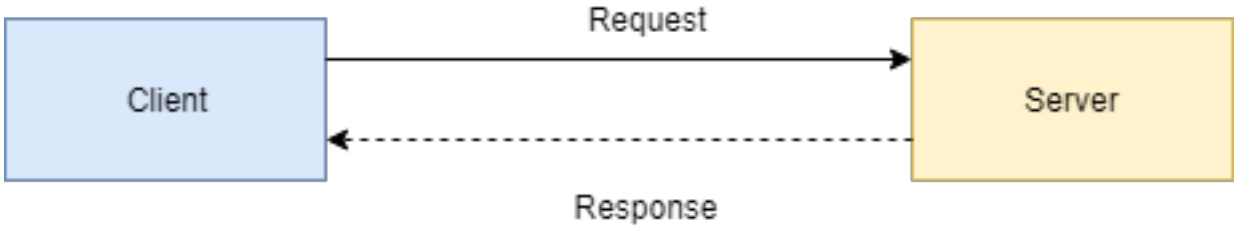

Figure 1: Client Server architecture

Furthermore, the system can be divided into three different subsystems: the presentation layer, the application layer and the data layer as we can see in Figure 2.

- The *Presentation Layer* provides the GUI of the system. This layer contains the mobile application and the web pages.
- The *Application Layer* contains the logic of the application,that receives the requests from the user, computes the best path to reach the appointment, checks the weather and the road conditions and executes the dynamic web pages of the web site.
- The *Data Layer* stores and maintains the data needed from the system to works properly, i.e. user's information and user's appointment information.

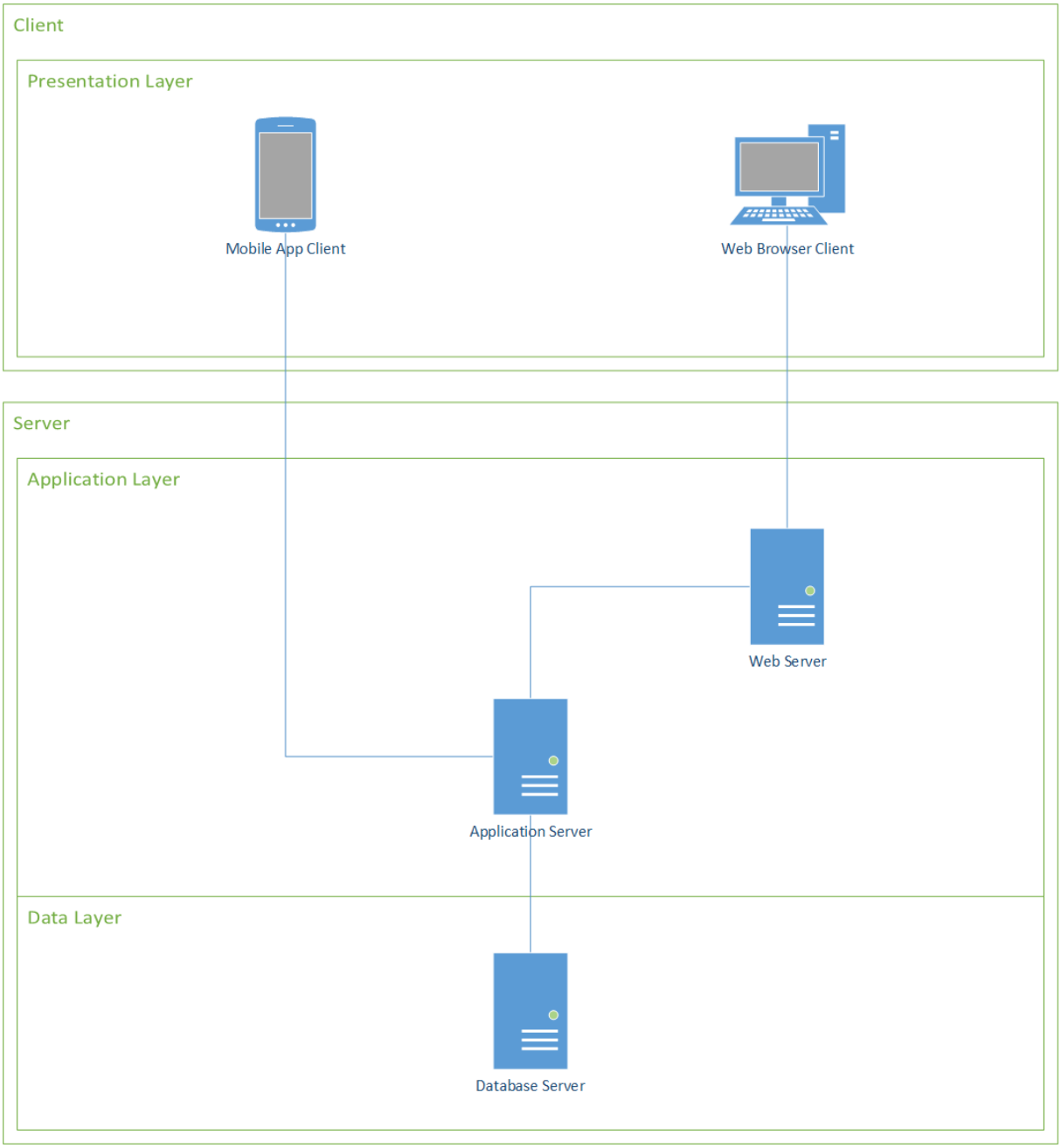

Figure 2: Overview of the system architecture

2.2 Component View

2.2.1 Overview

In Figure 3 is possible to see the high level components of the system and the interfaces used to connect one to another, where

- the *DBMS* provides the database and a way to retrieve data from it;
- the *Application Server* provides the main logic of the application;
- the *Web Server* provides the static pages and executes the dynamic pages of the web site.
- the *Mobile Application* is the mobile application used by a user with his/her smartphone.
- the *Web Application* is the application that runs on the user's browser.

2.2.2 Database View

The DBMS component provides a database and is DBMS for data storage and their management. It is possible to access the database only through the Application Server and an appropriate secure interface. For security and privacy reasons, data are encrypted inside the database. The Entity-Relationship diagram of the database is showed in Figure 4.

2.2.3 Application Server View

The *Application Server* contains the main logic of the application. It receives the user's request and interacts with the database to store and retrieves data. The *Application Server* as we can see in Figure 5 is composed of:

- **Authentication Manager**, it manages the request of a user to register or to login into the service. It can access to *Account Data Manager* to retrieves user's information from the database.
- **Profile Manager**, it manages the request of a user to update his/her profile. It can access the *Account Data Manager* in order to retrieves information in the database.
- **Account Data Manager**, it can access all the information about the user's account in the database.
- **Appointment Manager**, provides to the user the functionalities of creation / modification of appointments. It uses the *Path Calculator* to obtain the best path for the appointment and the *Appointment Data Manager* to stores and retrieves information.
- **Path Calculator**, it is responsible to compute the best path from the starting location defined by the user and the appointment location. To do so, it can access the *Additional Info Facade* to retrieves the user preferences, the weather and road informations. It needs also the *Google Maps API* to retrieves distance and time informations.
- **Weather Information Manager**, it manages weather information retrieving it from an external system via its API, showed in the diagram as *Weather API*.
- **Road Information Manager**, it manages road information retrieving it from an external system via its API, showed in the diagram as *Road API*.
- **Additional Info Facade**, it is a component that implements the *Facade Pattern*, in this way it is possible to reduce the coupling between the *Path Calculator* and the other interfaces from that needs to get information required.

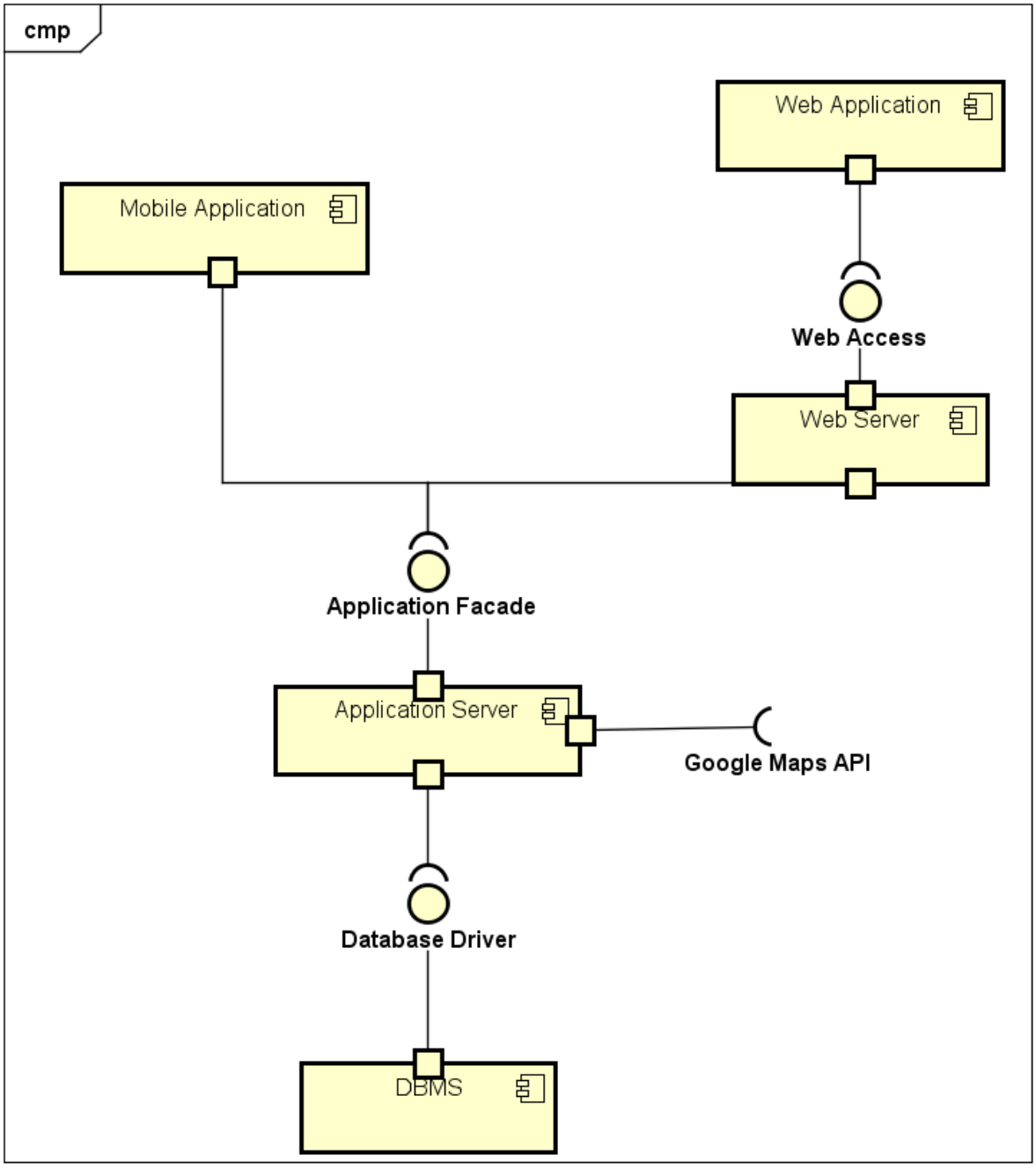

Figure 3: High level Component Diagram

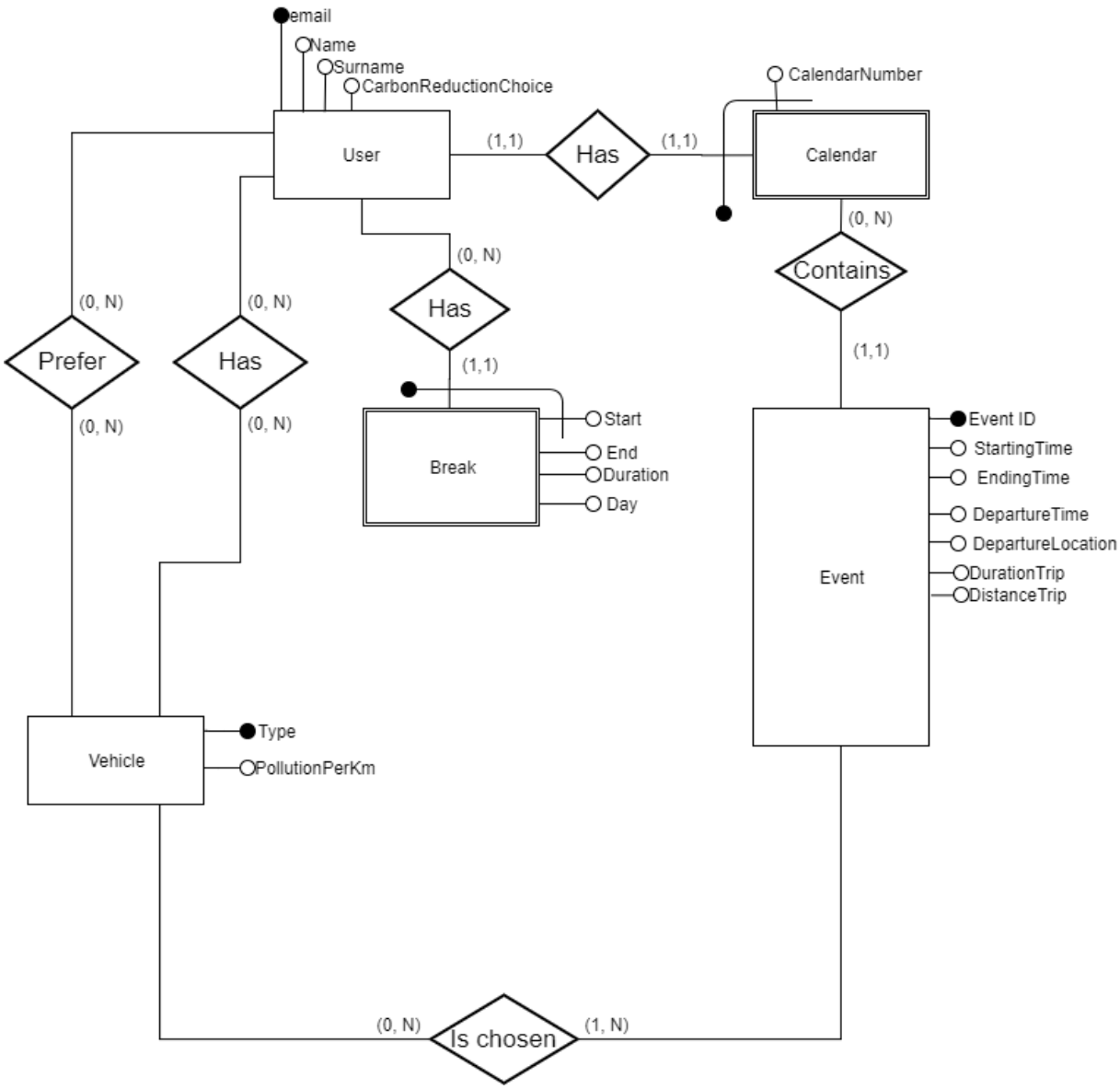

Figure 4: *Entity-Relationship Diagram* of the database

- **Travlendar+ Facade**, it is a component that implements the *Facade Pattern* and provides a common interface for both the *Mobile Application* and the *Web Server*.

2.2.4 Web Server

Since the presence of a *Web Application*, it is necessary a dedicated *Web Server* responsible to executes the web site's dynamic pages and provides the static pages to the user's browser. The *Web Server* interacts with the *Application Server* to get the proper information to fill up the pages. The *Web Server* also sends data from the user's browser to the *Application Server* to store inside the database.

2.2.5 Mobile Application

The *Mobile Application* is used by the user via its own smart device. The *Mobile Application* communicates directly the *application server* with a dedicated communication protocol. The component diagram of the *Mobile Application* is showed in Figure 6. The description of the components is the follow:

- **User View** is responsible of the graphical representation of the app and the interactions with the user.
- **GPS Manager** is responsible to interact with the GPS Module of the smart device.
- **DBMS**, is a physical view of the main database while storing only the current user's data. It is used by the *Controller* to notify the user when an event is about to start.
- **Controller**, is responsible to interact with the *Application Server* and link together the other components.

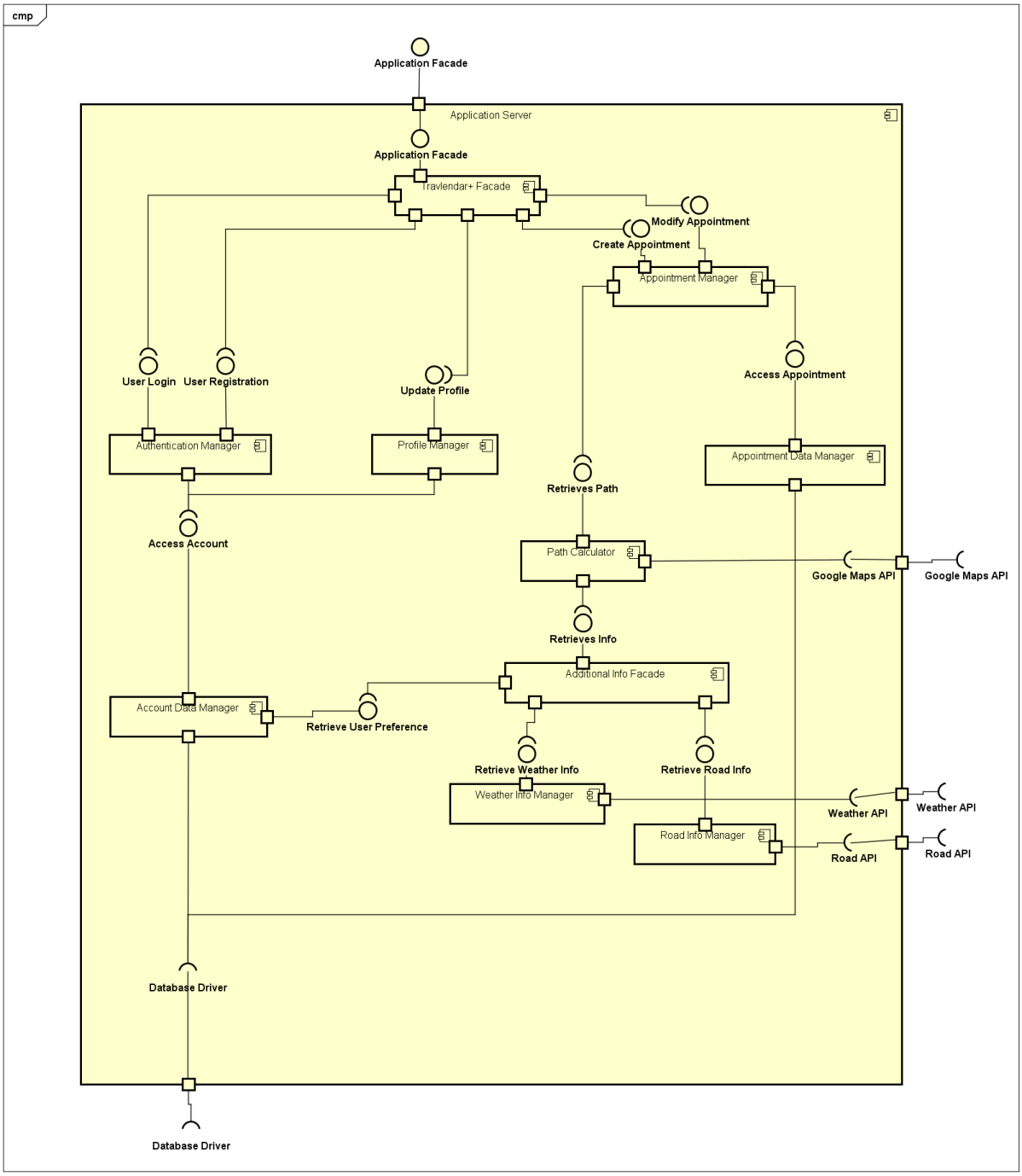

Figure 5: *Application Server* component diagram

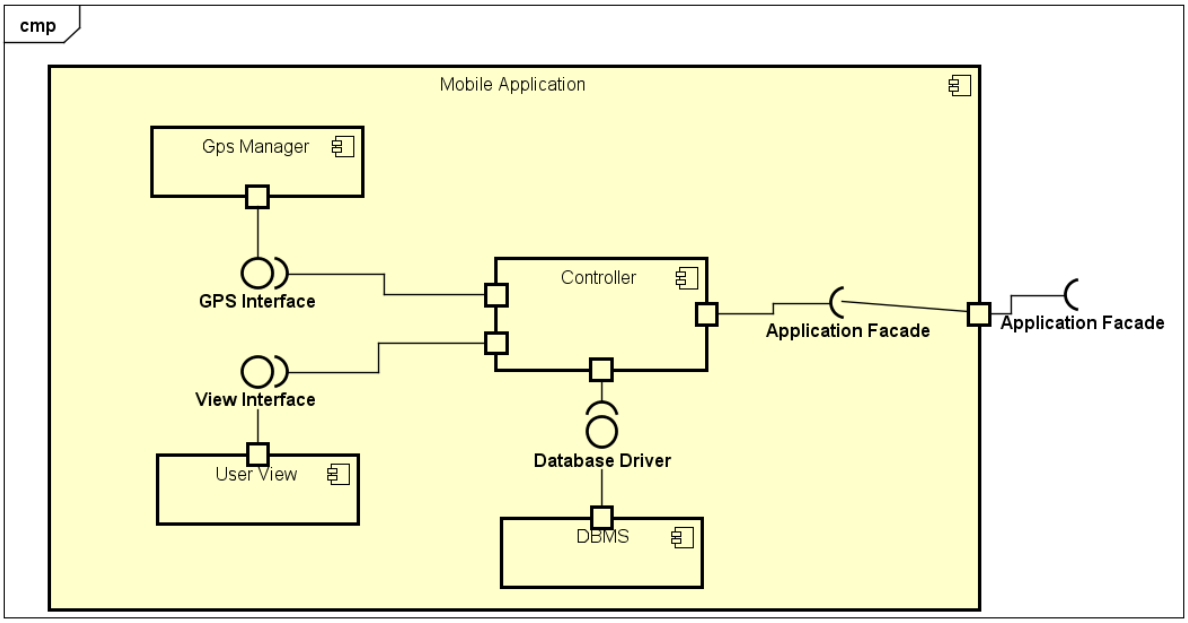

Figure 6: *Mobile Application* component diagram

2.3 Deployment View

The architecture chosen for Travlendar+ is a Four-Tier in Three-Layer one, where a high-level mapping layer to component is as follows:

- Presentation Layer: Mobile application/Web application and Web Server
- Logic Layer: Application server
- Data Layer: DBMS and Database

Note that the definition of layer is just a logic separation of the components that compose the system with the intent to better organise the code that needs to be developed, while a tier is a physical machine onto where the code is running.
To better understand the choice made a scheme is provided in Figure 7, where it's clear that there are three layers (the different coloured boxes) and four physical tiers (We consider the mobile and web application as a single tier, while obviously the code is written in the DBMS and not in the DB).

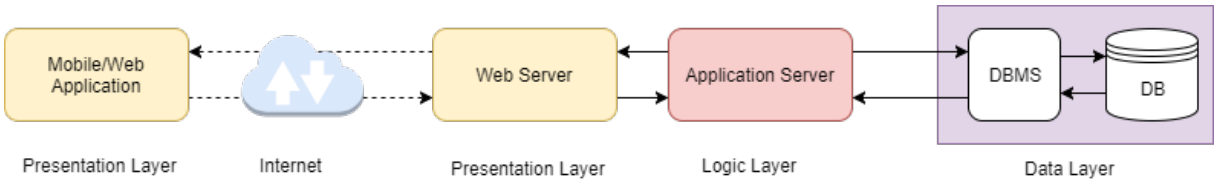

Figure 7: System's Tiers and Layers

Implementation choices
The technology chosen for the implementation on the system are mainly based on Java Enterprise Edition (JEE) since it offers a large number of tools and alternatives to develop multi-tier systems that need web based logic and storage and having at the same time the possibility of adding new functionalities in future, making the system more scalable.
Web Pages: The choice fell on JSP given the flexibility that a few snippets of Java code in a dynamic web page can provide.
Application Logic: EJB was the selected technology given that the system is developed mainly using JEE.
Application Server: GlassFish 5.0 has been chosen over other alternatives since it's an open source application server fully supported by Oracle. **Web Server:** GlassFish 5.0 was chosen again for coherence with the application server.
DBMS: MySQL was selected given that is supported by Oracle and is well known, making the amount of documentation available quite large. It was paired with InnoDB because it's the currently most used alternative and allows us to use foreign keys.

A complete overview of the technologies chosen can be seen in the decision flow diagram in Figure 8

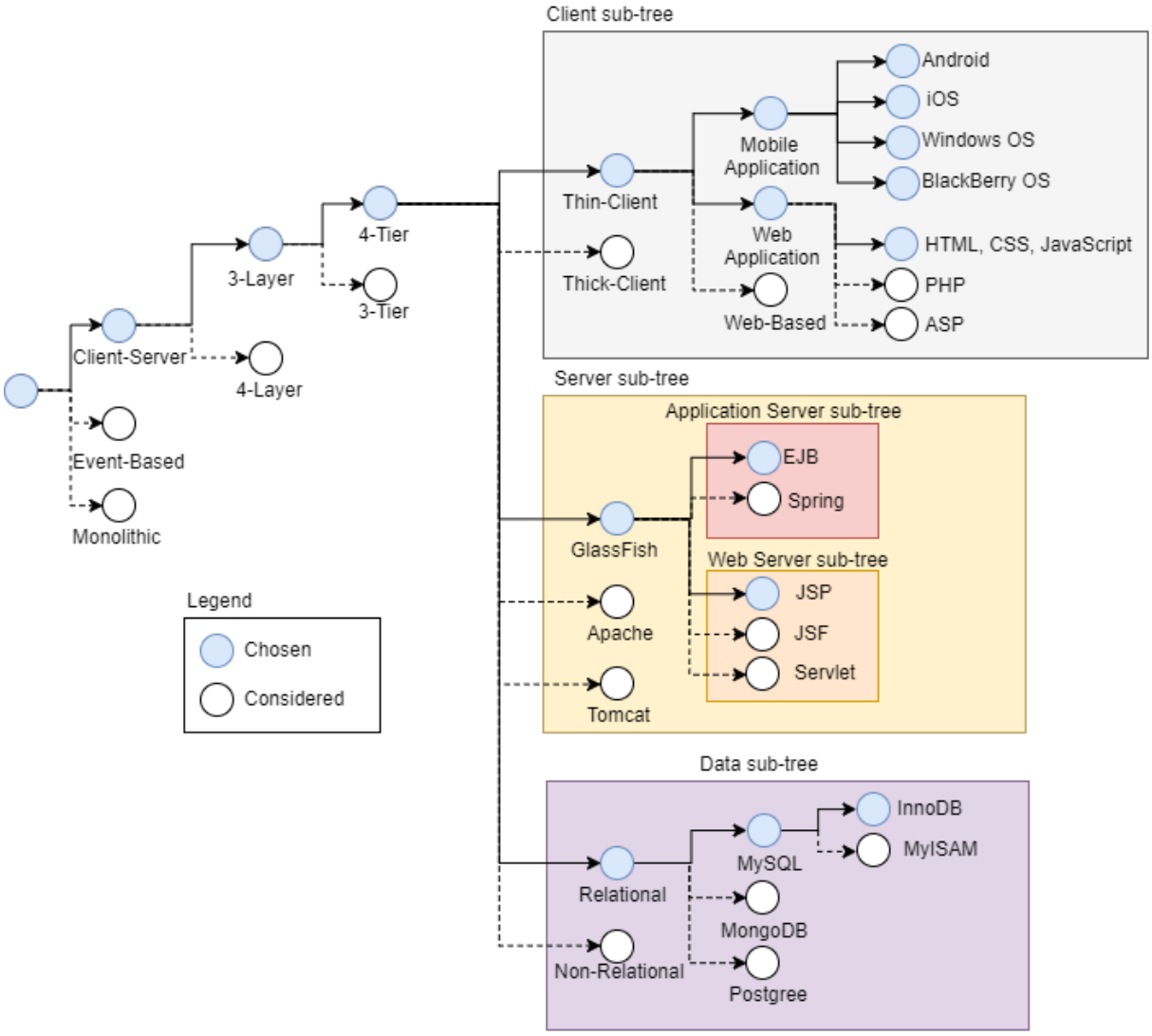

Figure 8: Decision Flow Diagram

2.3.1 Deployment Diagram

The deployment diagram can be seen in Figure 9, note that the components specified in Figure 6 and in Figure 5 can be seen in this representation of the mapping on concrete devices.

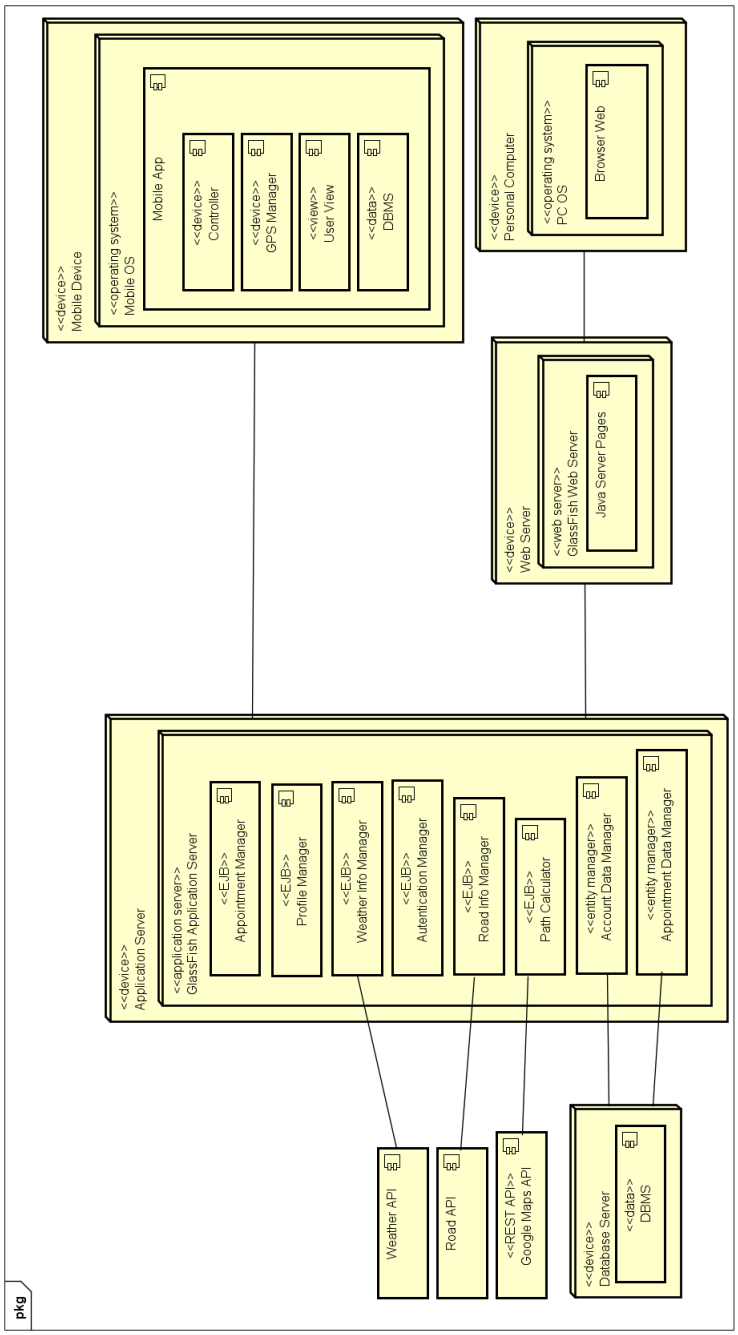

Figure 9: Deployment Diagram

2.4 Runtime View

In this section are represented the most important runtime views by using sequence diagrams that highlight the main interactions between the user and the components of the system for each analysed use case.
Note that each *Error Message* is just an abstraction of different errors with different codes for different situations, this was done in order to simplify the diagrams.

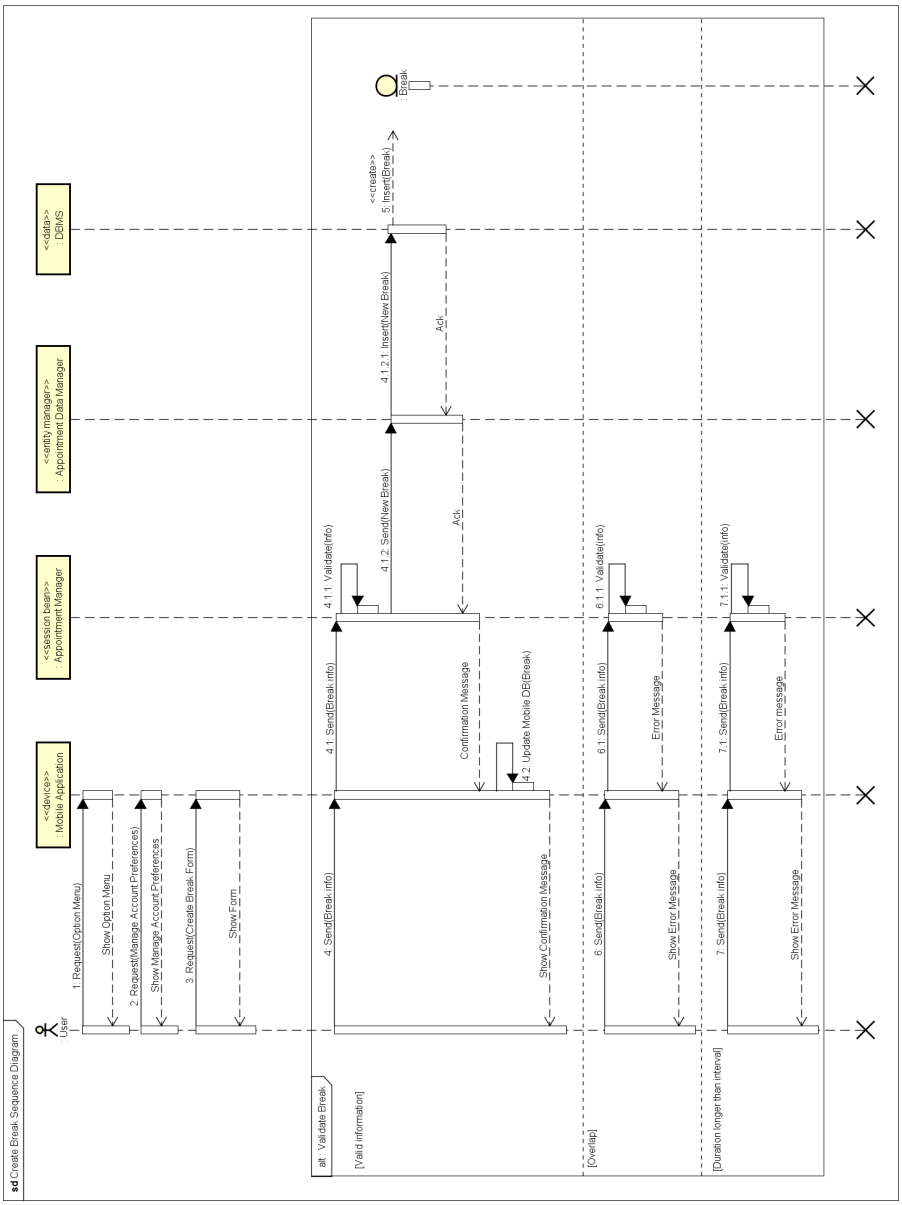

Figure 10: Diagram of the interaction between components in the *Create Break* use case, the *AppointmentManager* manages also breaks

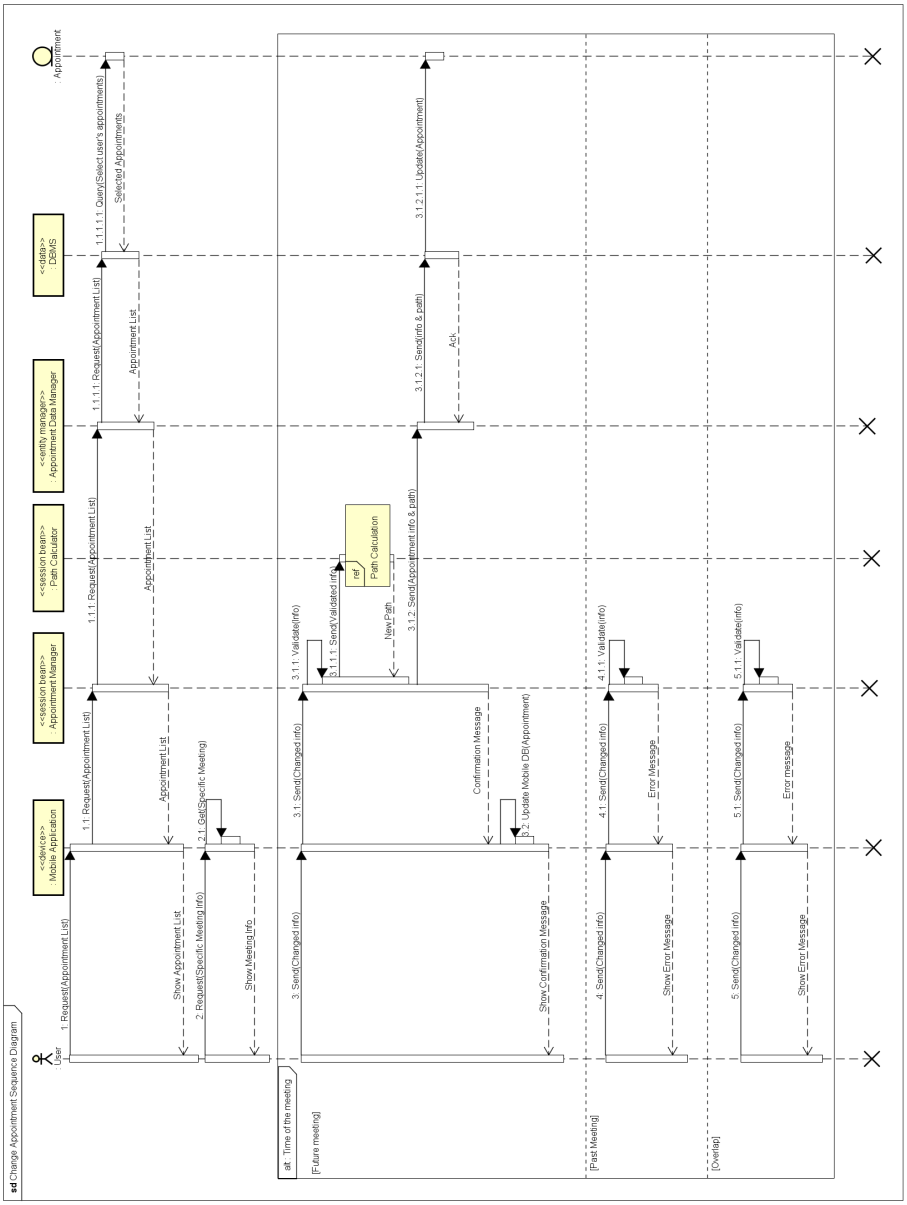

Figure 11: Diagram of the interaction between components in the *Change Appointment* use case, it should be noted that the internal interaction of the path calculator is better specified in Figure 14 in order to keep the diagram as clean as possible

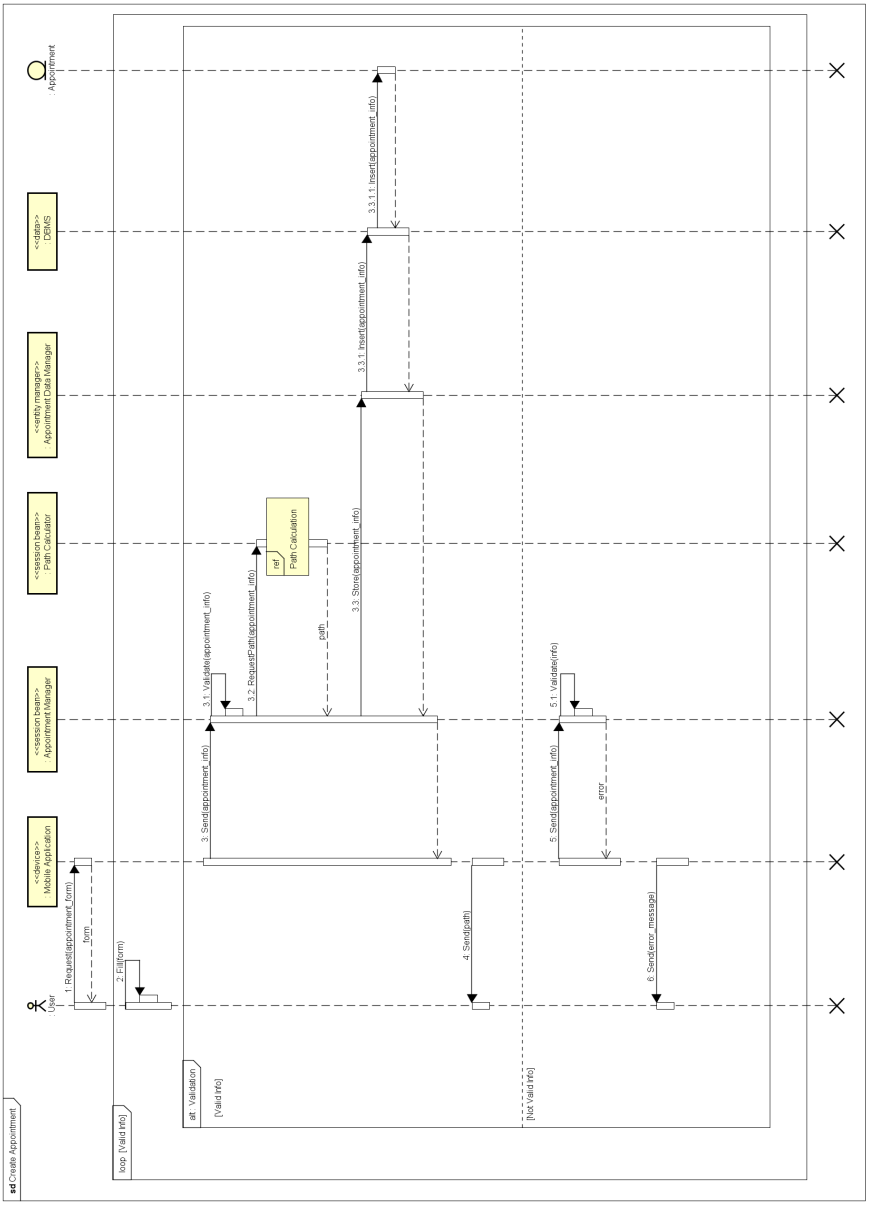

Figure 12: Diagram of interaction between components in the *Create Appointment* use case, the internal interaction of the path calculator is in Figure 14

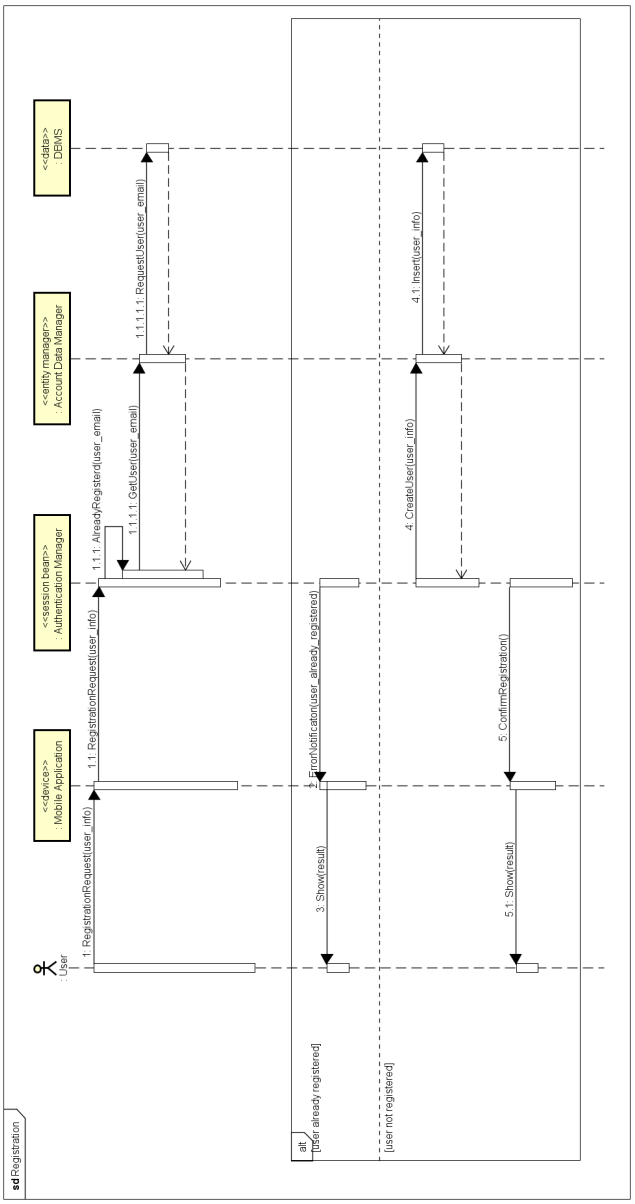

Figure 13: Diagram of interaction between components in the *Registration* use case

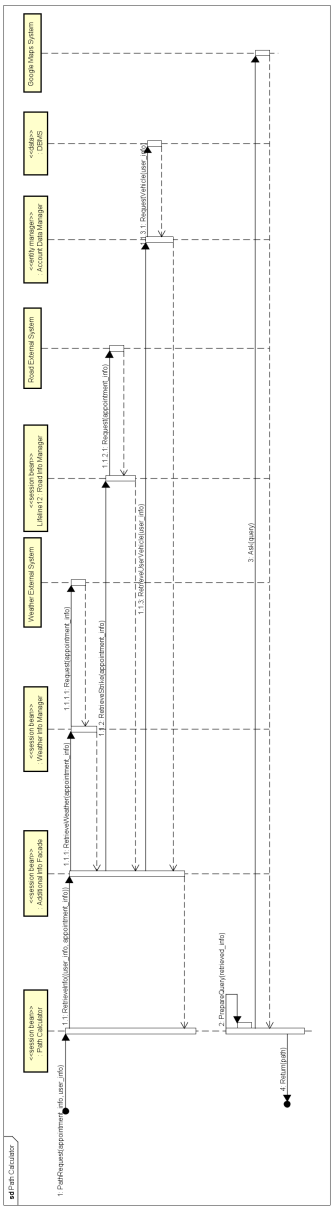

Figure 14: Diagram of the interaction between components during a calculation of a path.
The *Path Calculator* asks for information to the *Additional Info Facade*, and, based on this information, prepare the request to send to *Google Maps*

2.5 Communication Interfaces

2.5.1 Database Driver

This interface is used to allows the interaction between the *Application Server* and the *DBMS*.
There are two different components that interacts with this interface:

- Account Data Manager, for managing user's information.
- Appointment Data Manager, for managing appointments' information.

2.5.2 Application Facade

This interface provides a common point of access for both the *Mobile Application* component and the *Web Server* component.
It is provided form the *Travlendar+ Facade*.

2.5.3 Google Maps API

This interface can provide:

- a map of the path from the departure location to the appointment location.
- the ETA
- the path directions for each travel means.

The *Path Calculator* component builds the path from the information retrieved from this API.

2.5.4 Weather API

This interface provides a way to obtain the forecast of the appointment date. The *Path Calculator* component relies on this informations to take decision about the best path.

2.5.5 Road API

This interface provides a way to obtain information about possible strikes on the appointment date. The *Path Calculator* component relies on this informations to take decision about the best path.

2.6 Selected architectural styles and patterns

2.7 Other design decision

3 Algorithm Design

Introduction
Once the system has the couple time-place for both the departure and the arrival, it must take into account all the preferences of the user, weather conditions and public transportation strikes in order to formulate the best possible query that will be forwarded and handled by Google Maps.

4 User Interface Design

4.1 UserInterfaces

5 Requirements Traceability

5.1 requirements traceability

6 Implementation, Integration and Test Plan

6.1 implementation

7 Appendix

7.1 Effort Spent

7.2 References

