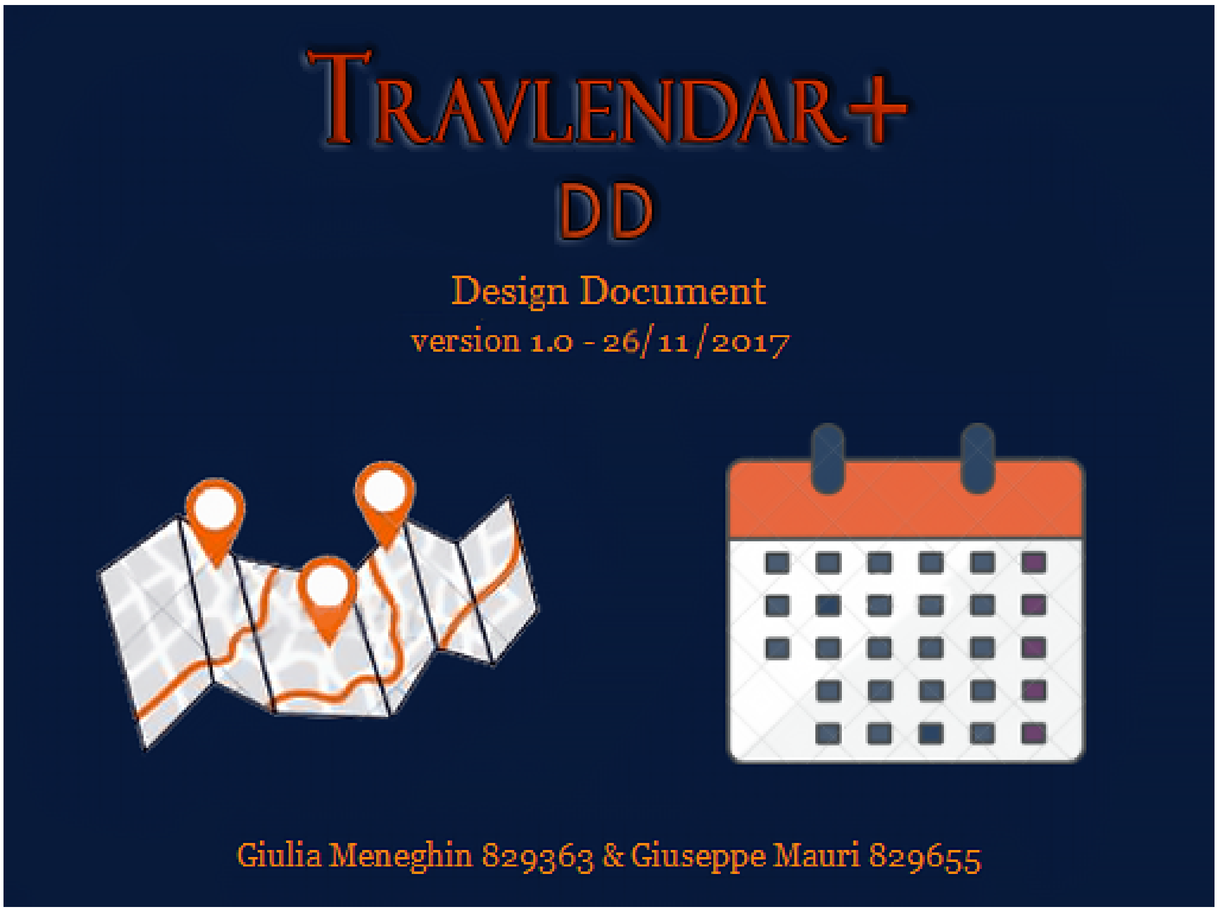
Politecnico di Milano
AA 2017-2018

Computer Science and Engineering
Software Engineering 2 Project



Deliverable:	DD
Title:	Design Document
Authors:	Meneghin Giulia & Mauri Giuseppe
Version:	1.0
Date:	26-November-2017
Download page:	`<https://github.com/Ciuse/MauriMeneghin>`
Copyright:	Copyright © 2017, Meneghin Giulia & Mauri Giuseppe – All rights reserved

Contents

1 Introduction

1.1 Purpose

1.2 Goals

(G1) Allow the user of the application to create a personal account and modify his information.

(G2) Allow the user to change the current day with an interactive calendar.

(G3) Allow the user to add a dish to a specific mealtime of the selected day(Breakfast, Lunch, Snack, Dinner), setting the eaten quantity.

(G4) Allow the user to search a dish.

(G5) Allow the user to scanning a bar code with the camera application.

(G6) Allow the user to create a new dish with a specific image and a lists of ingredients.

(G7) Allow the user to add a symptom in a specific day, setting the intensity, frequency, mealtime occurrences.

(G8) Allow the user to delete the inserted dishes.

(G9) Allow the user to delete the inserted symptoms.

(G10) Allow the user to visualize the percentage of symptoms occurrences and ingredients eaten in a days range (weekly or monthly).

(G11) Allow the user to visualize the correlation between the symptoms occurrences and the eaten ingredients.

(G12) Allow the user to add a new treatment specifying the diet and medical cure.

(G13) Allow the user to remove a treatment.

(G14) Allow the user to visualize the in progress treatments and the completed ones.

(G15) Allow the user to change the profile image.

1.3 Functional Requirement

(G1) Allow the user of the application to create a personal account with an email and password, or with an external services (Google, Github, Twitter).

- The user should be able to register through the mobile application.
  The user must provide user-name, password, e-mail, and a valid address that will be used as the default position.

- The user must insert the transport means preferences.
  Then he can accept or modify the default limits about them.

- Finally, he can accept or modify the default set of “Type of appointment”.

- The user must be able to modify all the option even after he finished the registration process.

1.4 Scope

Bealthy is an application that allows the user to enter the dishes they consume and monitor the symptoms they experience on a daily basis. The entry of dishes can be done by: searching a predefined list of foods known to our application, scanning a barcode with the camera, or manually entering the information of the dish. The purpose of the application is to collect the data entered by the user, process it and reorganize it in the form of a graph to show the user the correlation between the symptoms experienced and the individual ingredients consumed. The application implements three types of graphs: The first consists of showing which symptoms/ingredients have a higher percentage of the total in a specific period of days. The second type shows which ingredients affect a particular symptom in the selected time period. The third shows of an ingredient what is the likelihood of causing one or more symptoms. In addition, the system allows the user to enter his or her medical treatments that he or she is following and shows past treatments. The system compares the previous month’s symptom data with that of the days under treatment to show the effectiveness of the treatment taken.

1.5 Acronyms

- *API:* Application Programming Interface.

1.6 Revision history

– Version 1.0

1.7 Document Structure

1.7.1 Introduction:

The introduction to this section describes the main features of the design document. This section highlights more technical aspects that were not dealt with in the RASD document. We can distinguish different subsections of this document:

1.7.2 Architecture Design:

– Overview: The top-level components of our application are described.

– High level components and their interaction: This section focuses on how the various components interact with each other.

– Component View: This section provides a more detailed view of the application components. We will use the component diagram which will show how the components of our application are connected together to form larger components. The system structure is shown.

– Deployment View: This section shows how software components are distributed over the hardware resources available on your system. We will use a Deployment Diagram that statically describes our system in terms of hardware resources, called nodes, and relationships between them.

– Runtime view: In this part, sequence diagrams will be used to describe how components interact to perform specific tasks typically related to usage cases.

– Component interfaces

– Selected architectural styles and models: This section explains the architectural choices made during the implementation of the application.

– Other design decisions

1.7.3 System interfaces

1.7.4 Software interfaces

1.7.5 Algorithms Design:

1.7.6 User Interface Design:

This section presents examples of mockups and user experiences.

1.7.7 Requirements Traceability:

This section explains how decisions taken in the RASD are related to design elements.

1.7.8 Implementation, Integration and Test Plan:

This last section proposes the order in which we plan to implement the sub-components of our system and the order in which we plan to integrate these sub-components and test the integration.

2 Architectural Design

2.1 Overview

2.2 High level components

2.3 Component view

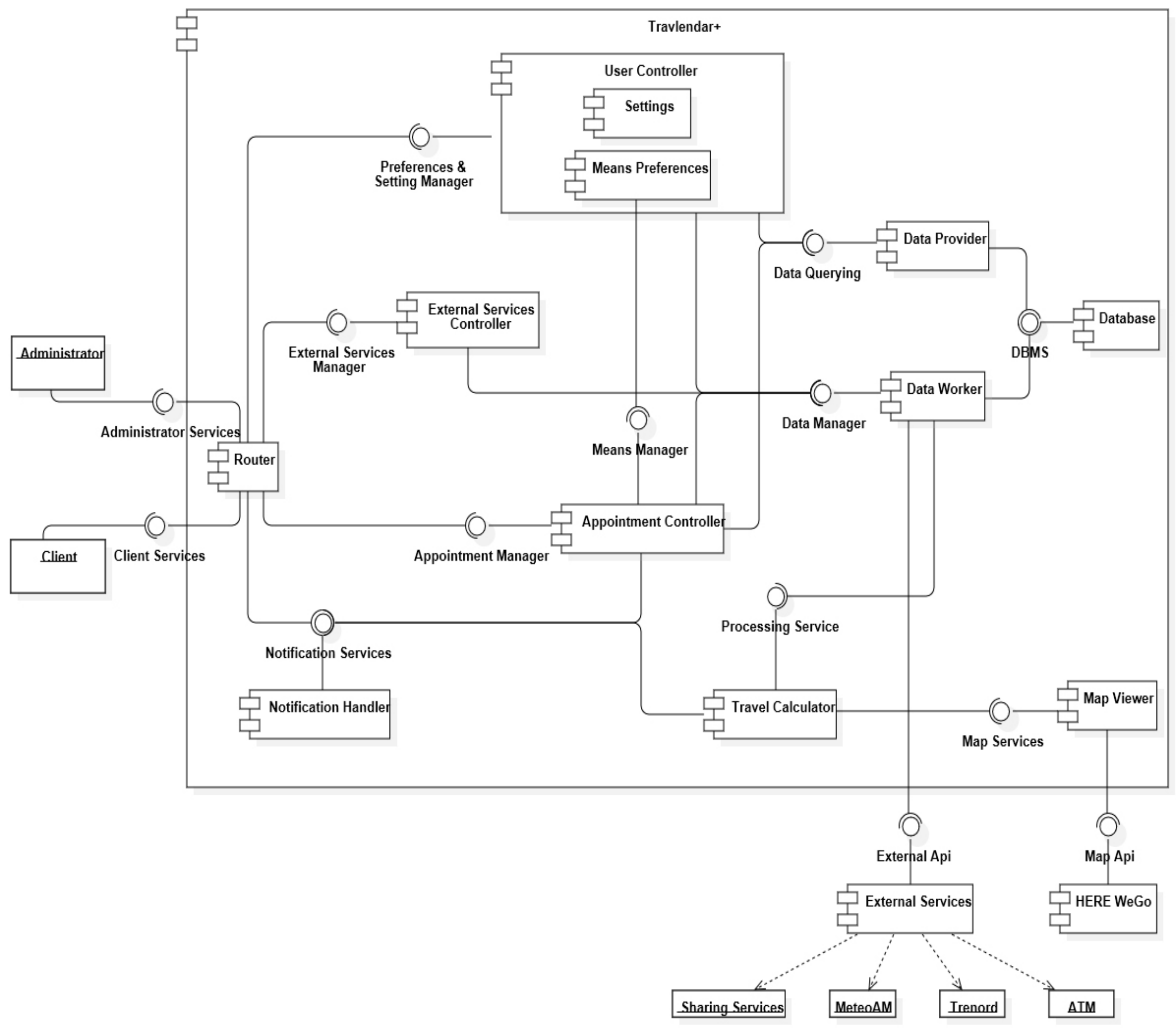


Figure 1: Component Diagram

- Client: the client's device (mobile app).
- Administrator: the administrator's device (mobile app).

2.4 Runtime view

In the following Sequence Diagram for simplicity and clarity, some steps have been omitted in the communication between the components external to the main system and the router; as the main aspects of communication, between the User and the Travlendar system, have been dealt with in the RASD document.

2.4.1 Sequence Diagram 1

This Sequence Diagram deals with the addition of an appointment by the customer. The request is taken on board by the router, which communicates with the appointments component and will show the user the form to fill in the data concerning his new event.
Subsequently, after the user has inserted the various information, the Appointment Controller component will check that it does not overlap with an existing one, in case the Notification Manager will send the user an error.
If the appointment is valid, the data worker will be asked to save the new appointment on the database and the latter will be responsible for providing the travel calculator with all the data necessary to calculate the trip from the previous position to that of the appointment, with the means and preferences that the user has selected.
If it is possible to solve the calculation, the travel itinerary will be saved and the user will receive a confirmation message; if not, it will be notified with a warning.

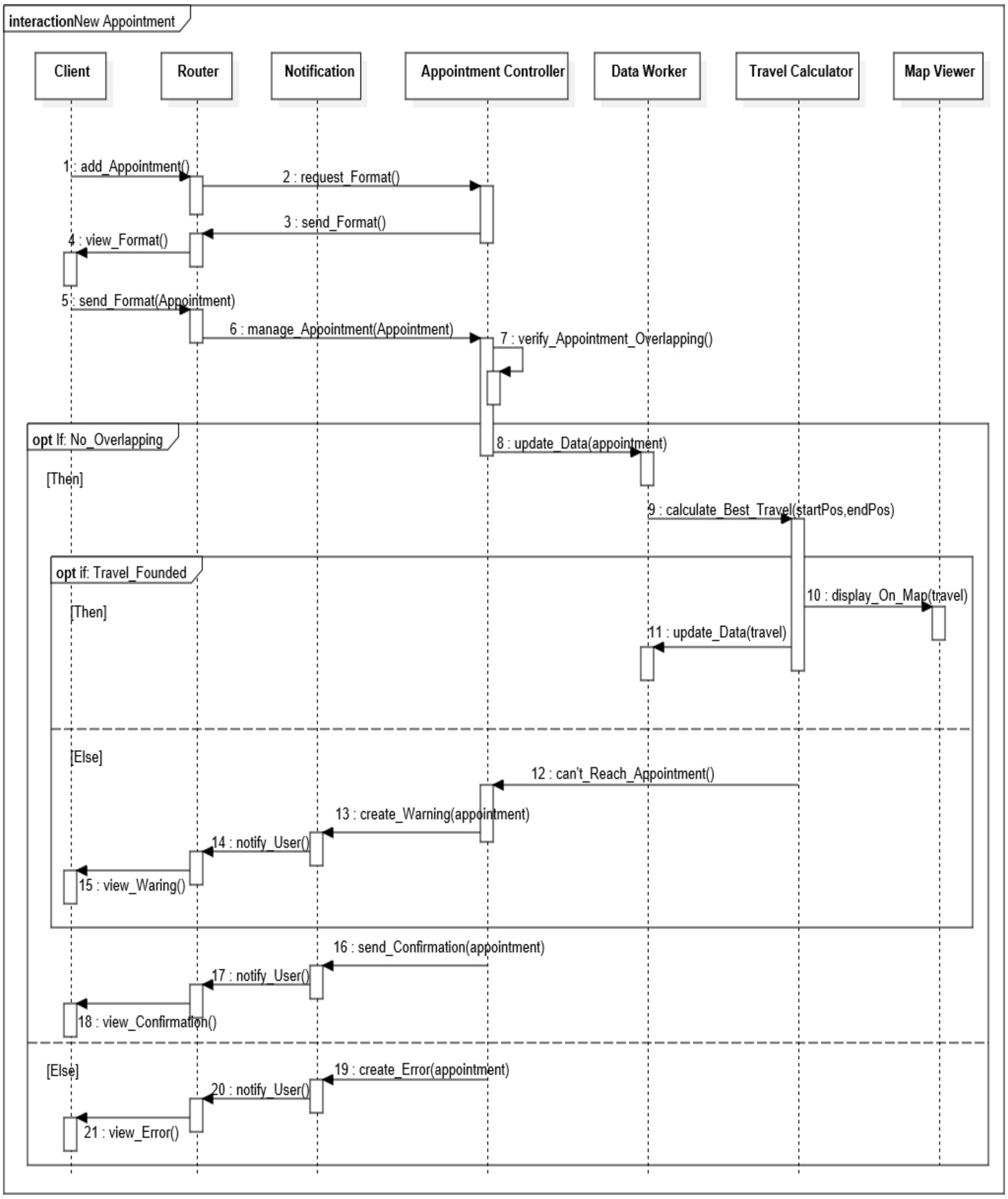
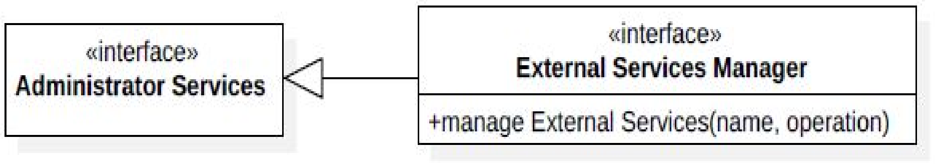


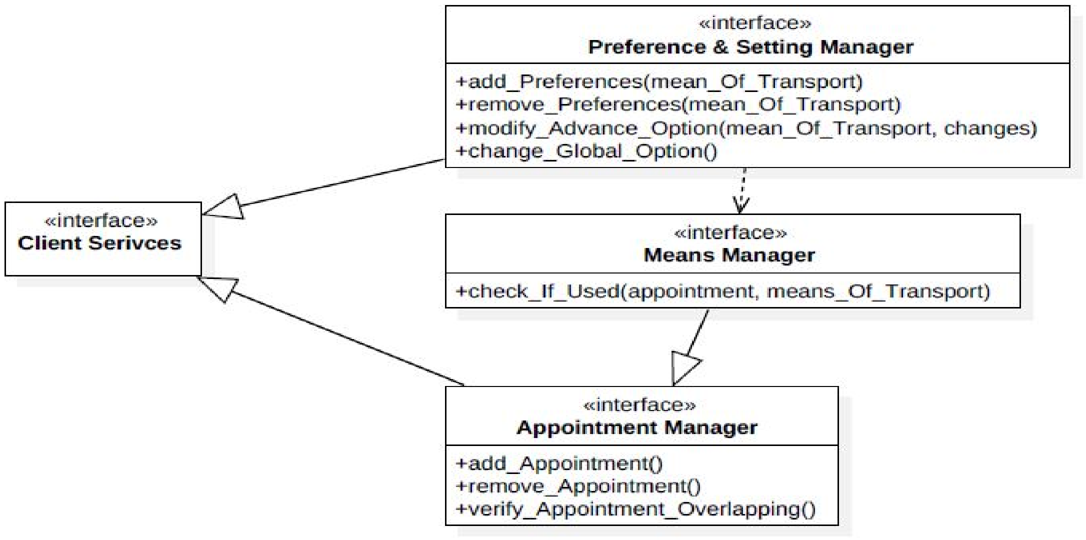
Figure 2: Sequence Diagram 1

2.5 Component interfaces

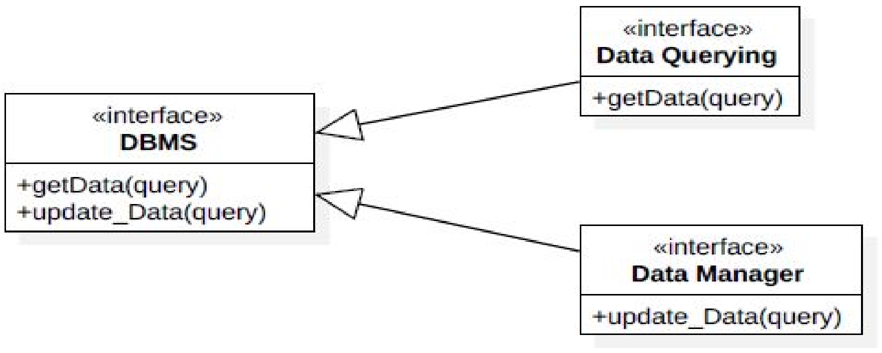
2.5.1 Administrator Services


«interface»
Administrator Services
«interface»
External Services Manager
+manage External Services(name, operation)

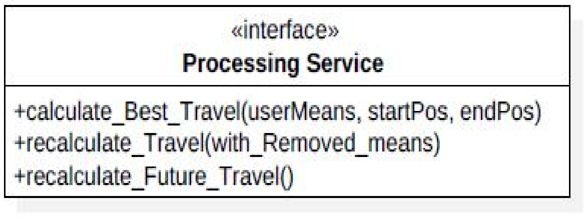
2.5.2 Client Services


«interface»
Preference & Setting Manager
+add_Preferences(mean_Of_Transport)
+remove_Preferences(mean_Of_Transport)
+modify_Advance_Option(mean_Of_Transport, changes)
+change_Global_Option()
«interface»
Client Serivces
«interface»
Means Manager
+check_If_Used(appointment, means_Of_Transport)
«interface»
Appointment Manager
+add_Appointment()
+remove_Appointment()
+verify_Appointment_Overlapping()

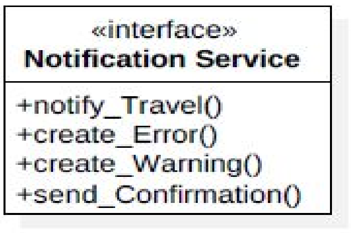
2.5.3 DBMS



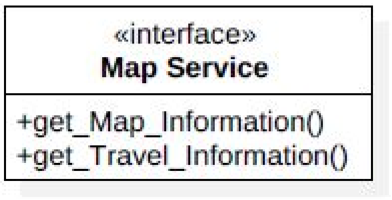
2.5.4 Processing Service


«interface»
Processing Service
+calculate_Best_Travel(userMeans, startPos, endPos)
+recalculate_Travel(with_Removed_means)
+recalculate_Future_Travel()

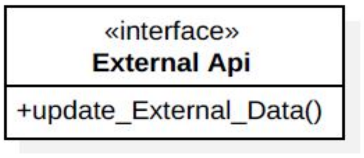
2.5.5 Notification Service

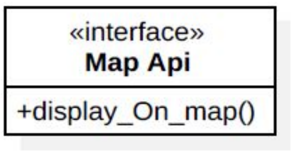


2.5.6 Map Service



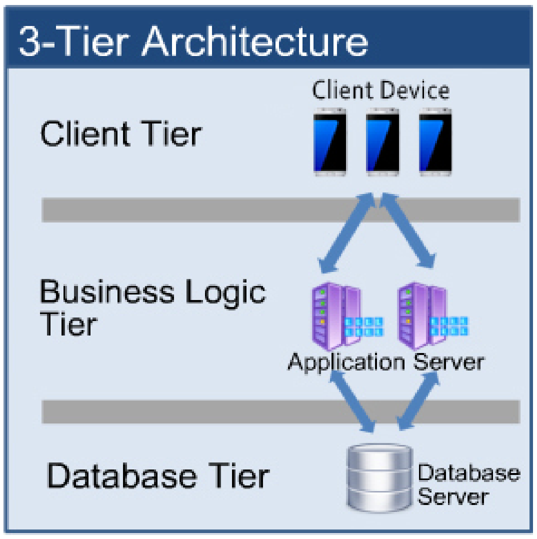
2.5.7 API Interface





2.6 Styles and patterns

2.6.1 Overall Architecture



Our application will be divided into 2 tiers: (fat client)

1. Database tier (DAL: Data Access Layer)
2. Business Logic tier (BLL: Business Logic Layer) (mobx)
3. Client tier (interface to BLL)

2.6.2 Design decisions

2.6.3 Design patterns

3 Algorithm design

3.1 Brief description of the best travel calculation algorithm

3.2 Software System Attributes

3.2.1 Reliability

The users age range is 16-40 years old, so the system will be designed to be always reliable:

- On workday, in the early morning (7-10) and in the late afternoon (16-19);
- On weekend, from the midday to middle evening (11-16) and in the evening (19-24);

Because during the working day the user will use the application mainly for register his working or studying appointment and some personal engagements.
While in the weekend the user will register his leisure events and night friends meeting.

3.2.2 Availability

The system must guarantee a 24/7 service. Very small interrupt of the service during the day will be acceptable and tolerated. But when an issue occurs, the system must respond correctly after a maximum of 3 user attempts.

3.2.3 Security

Users credentials and external services account will be cryptate and stored in a reserved and protect area of the system database. Also, the privacy information about the user movement and his localization must be all encrypted and totally protected.

3.2.4 Maintainability

Our system uses many external APIs, so the maintainability of our software is very much dependent on this factor. Our system must always be updated with external API interfaces, and it must always be able to interpret and exploit the data it obtains from them. Generally, there should not be too much invasive maintenance processes, as it uses very popular APIs used by many other software, whose changes are often minimal and well documented.

3.2.5 Portability

The system in terms of portability shall be very flexible. The application logic and the system interfaces are abstractly separated; so, the application porting consist only in the re-adapting of the user interface with the new operating system. Also, could be necessary to reimplement some of the system interaction with the external services.

4 User Interface Design

5 Implementation,Integration and Test Plan:

5.1 Elements to be integrated

5.2 Integration Testing Strategy

5.3 Component Integration

5.4 Used Tools

– StarUml 2.8.0

– Miktex 2.9.6361

– Texmaker 5.0.2

– DeepL

– GitHubDesktop 1.0.6

– AdobePhotoshop CC 2017

– Power Point