**Travlendar+ project Andrea Mafessoni,
Andrea Mazzeo, Daniele Moltisanti**

POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Andrea Mafessoni, Andrea Mazzeo, Daniele Moltisanti
Version:	1.0
Date:	29-October-2017
Download page:	https://github.com/AndreaMazzeo289/MafessoniMazzeoMoltisanti.git
Copyright:	Copyright © 2017, Andrea Mafessoni, Andrea Mazzeo, Daniele Moltisanti – All rights reserved

Contents

List of Figures

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). Aim of this paper is to exhaustively describe the different features and functionalities of the system, specifying goals and constraints of the overall project. The document contains a detailed picture of the system behaviour, both from the point of view of the user experience, in terms of expected use cases and offered features, and with a focus the internal structure of the application, in terms of functional and non-functional requirements. This document is intended to be read from all the stakeholders of the project and from any actual or future developer who will need to work properly on it.

1.2 Scope

The application intended to be developed is a mobile app for Android smartphones named Travlendar+. The application aims to provided a calendar-based system, in wich the user is allowed to insert and check his daily appointments and is supported during the travels to reach the provided meeting locations. The app is required to be more than a simple virtual calendar: it has to autonomously manage the different travel alternatives and collect information about external weather condition and availability of public transport in order to provide the user with a detailed schedule of his daily trips. The system must require the user to insert only the essential data for the appointment creation and must take care of everything concerns the travel organization, giving at the same time to the user the possibility of arrange differences travel preferences and switch between the possible travel alternatives. Plus, it must be open to advanced settings, allowing the user to create flexible and repeatable appointments or offering the functionality of adding alerts to remind each event. Travlendar+ also aims to implement a ticket-manager system: trough the application it must be possible to buy public transport tickets and view them when needed.

1.3 Definition, Acronyms, Abbreviations

1.3.1 Definition

- **Appointment**: Event scheduled by the user at a specific place, date and time.

- **Alert**: Reminder associated to a specific appointment that will ring at a given time.

- **Daily view**: specific layout of the calendar that shows all the appointments day by day.

- **Monthly view**: specific layout of the calendar that shows the different days in the classic monthly configuration.

- **Weekly view**: specific layout of the calendar that shows all the appointments week by week.

- **Daily schedule**: app function that shows the scheduled itinerary for a given day, specifying all the movements to make to reach the different destinations.

- **Movement**: Atomic shift from a place to a different one with a single transport mean.

- **Travel**: List of movements associated to an appointment to reach the event location on time.

- **Unreachable**: Appointment impossible to reach on time in the condition expressed by the user.

- **Repeatable**: Appointment required to be scheduled by the system more than once, in different dates expressed by the user.

- **Flexible**: Appointment required to be scheduled by the sistem in any free time inside a time interval allotted by the user.

1.3.2   Acronyms

- **RASD**: Requirement Analysis and Specification Document.

- **API**: Application Programming Interface.

1.3.3   Abbreviations

- **Dn**: n-Domain assumption

- **Gn**: n-Goal

- **Rn**: n-Functional requirement

1.4   Document structure

This RASD is composed by 7 parts:

1. The first part of RASD document is an introduction to the problem. The base information needed to understand the project scope are given in this section.

2. The second part consists of an overall description of the system. Are described the characteristics that reguard the user, the application dependencies and constraints.

3. The third part is composed by the specific requirements identified. The first version of user interfaces is showed and the APIs that are used. functional.

4. In the fourth section are presented four scenarios that show all application functionality.

5. The fifth part is entirely composed by the UML diagrams that model the system in details, starting from use case, sequence diagram and activity diagram.

6. The sixth part is dedicated to Alloy model of the system.

7. The last section shows the effort spent to make RASD.

2   Overall Description

2.1   Product perspective

The product we will develope is a software application for smartphone implementing Android as operative system. The app will require the user to be registered in order to access to all the funtionalities. It will also need a stable internet connection in order to connect to different external system providing information about weather and transport means. The application runs locally on the smartphone but users data are also stored in the the system database.
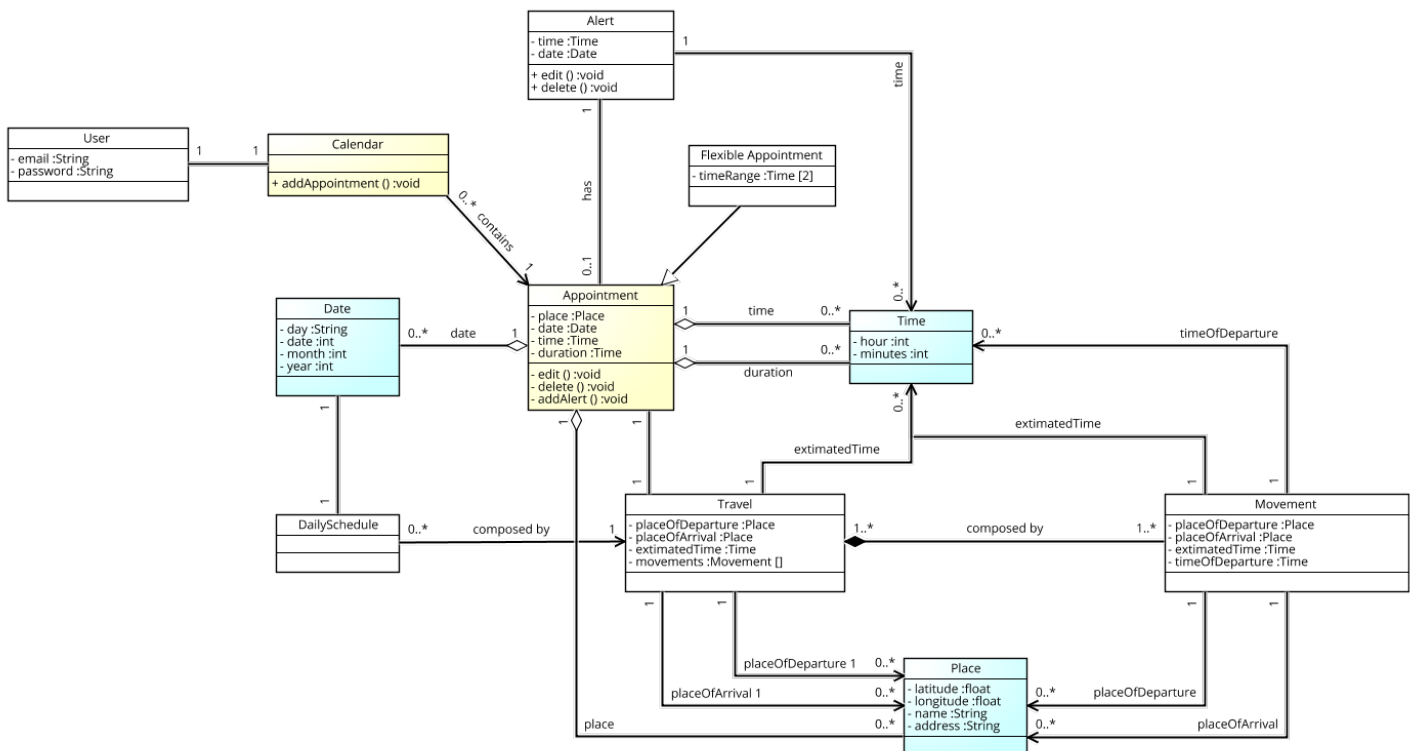
2.2   Product functions

This application aims to provide a smart and user-friendly appointment-manager system, which allows the user to create and keep track of several appointment on a personal virtual calendar. Plus, the app is able to schedule the travels to reach the different appointment destinations on time in the fastest and most confortable way, according to the preferences expressed by the user and the different weather and public transport condition. Finally, the application offers the possibility of easily buy tickets for the different trips, and to view them when needed.

2.3   User characteristics

We recommend this application to a everyone who wants to easily manage his appointments without any loss of time. The app is designed to be as user friendly as possible, so the user is not forced to have a specific knowledge of the system to be able to benefit of the service. It can be used by a business man with a calendar full of work meetings or just by an ordinary student to schedule his study breaks. Furthermore, the possibility of buying tickets suits even simple travelers who need to manage their trips.

2.4   Dependencies

- The application requires a stable internet connection, by Wi-Fi or mobile network.

- The application makes use of GPS localization to access to the user position.

- Based on the user choice, the application may require a connection to an existing Facebook or Google account.

- The application makes use of external APIs to achieve information about maps, public transport and weather condition.

- The application depends entirely on the ATM and Trenord public transport Payment Service for everything concerning the tickets purchase.

2.5   Constrains

- The application requires the user to be registred and logged into the system to work properly.

- The application works only on smartphones that implement Android version 4.0.3 or later.

- The application is in beta version and works only inside the urban area of Milan.

- 30 Mb(?) of storage memory must be available on the device in order to correctly install the application.

3 Specific Requirements

3.1 External interface requirements

3.1.1 User interface
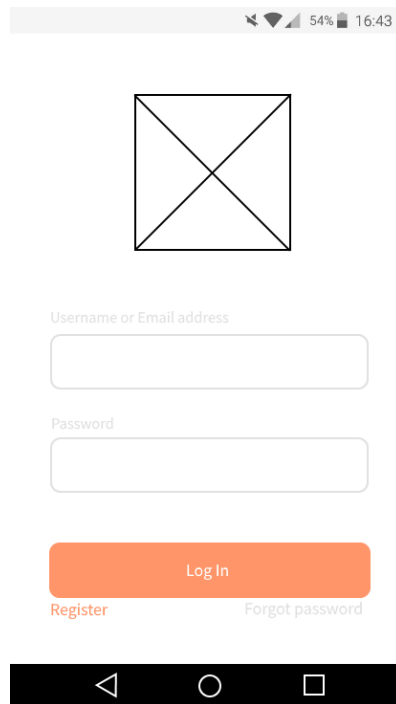
Figure 1: Login page
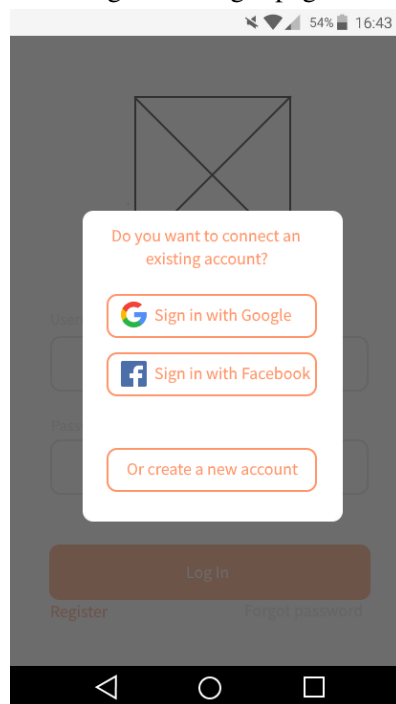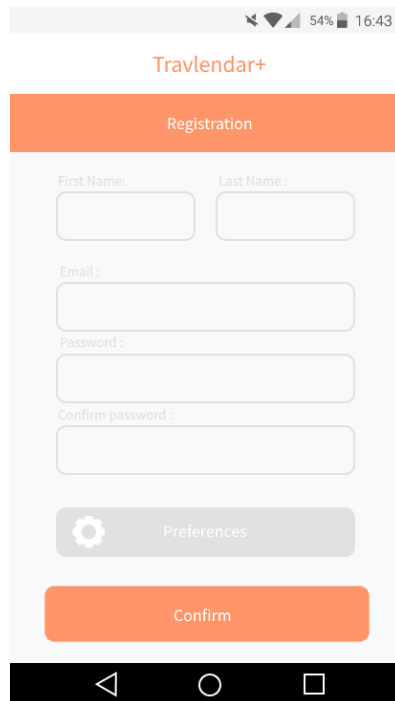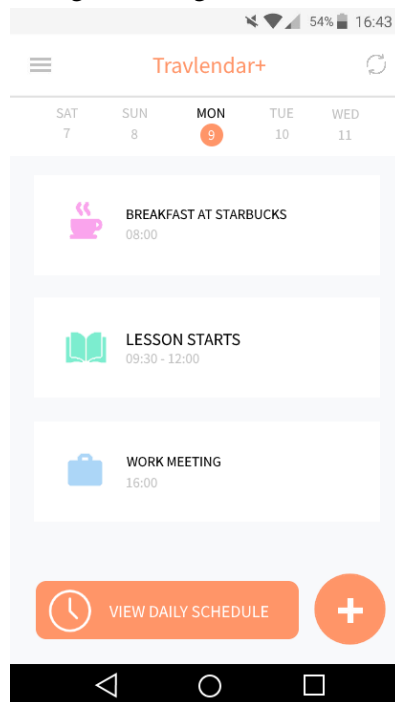
Figure 2: Connection of an existent account

Figure 3: Registration form

Figure 4: Daily view of calendar
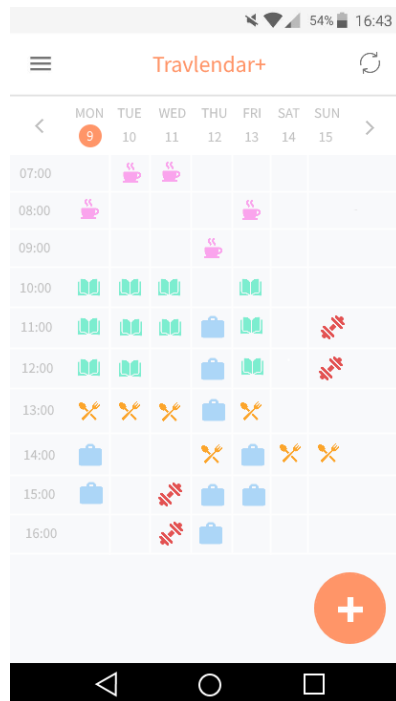
Figure 5: Weekly view of calendar
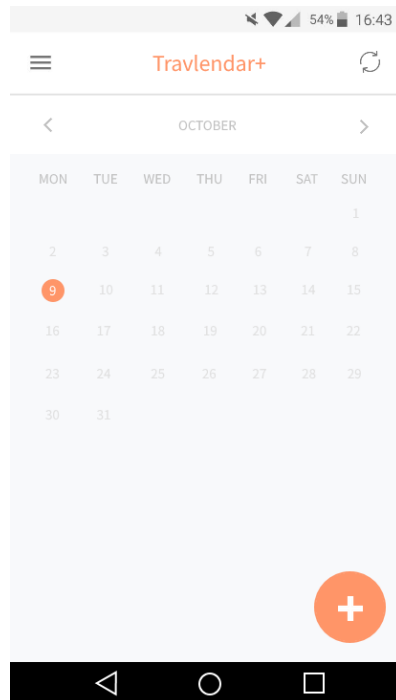
Figure 6: Monthly view of calendar
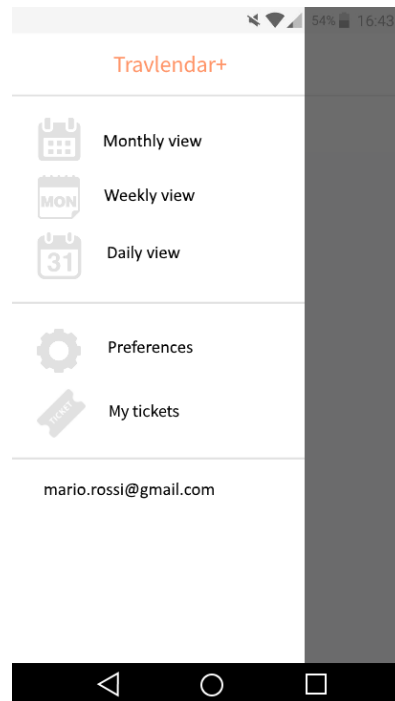
Figure 7: Menu
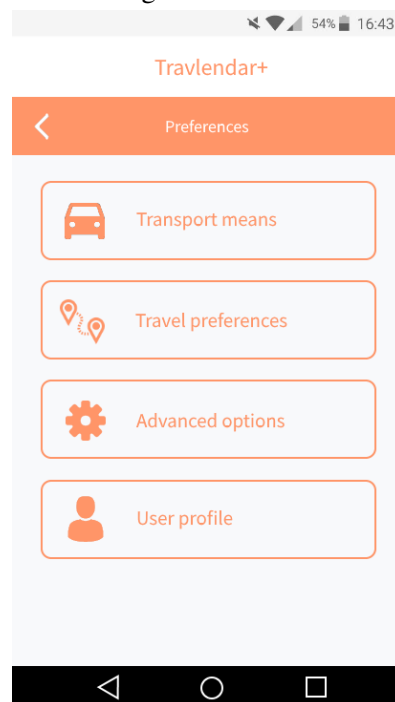
Figure 8: Preferences menu
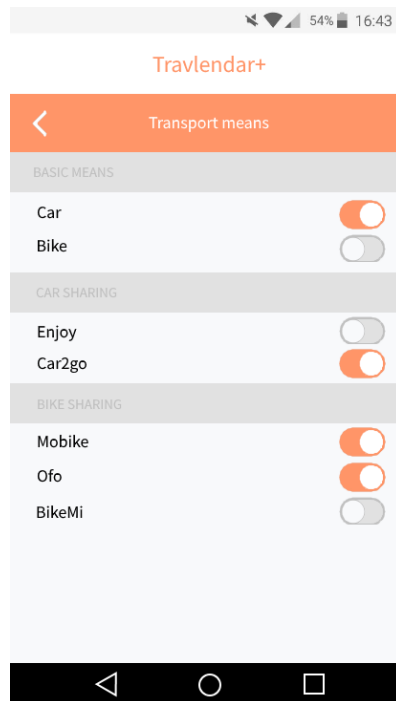
Figure 9: Transport means settings
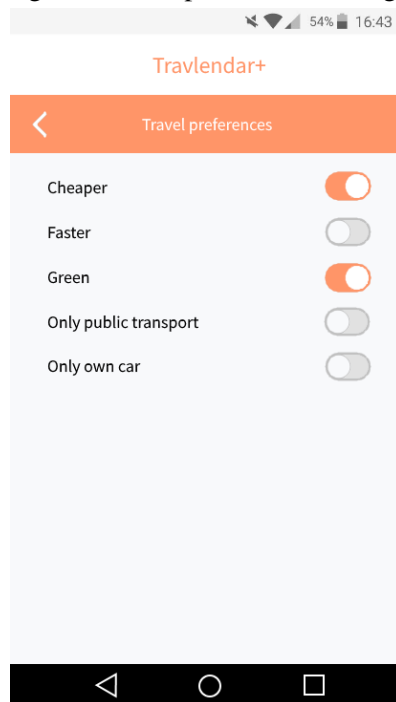
Figure 10: Travel preferences
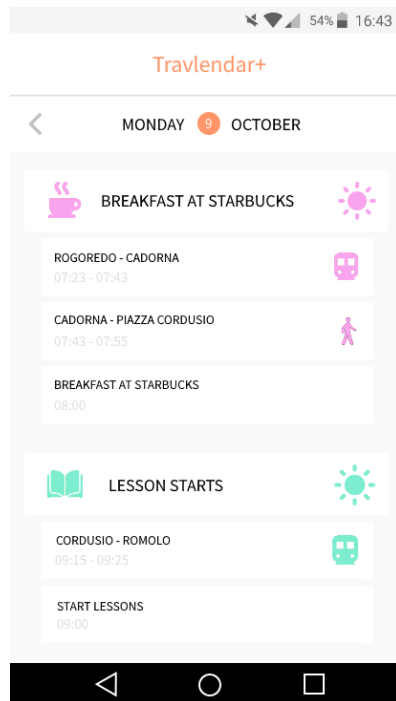
Figure 11: Daily schedule
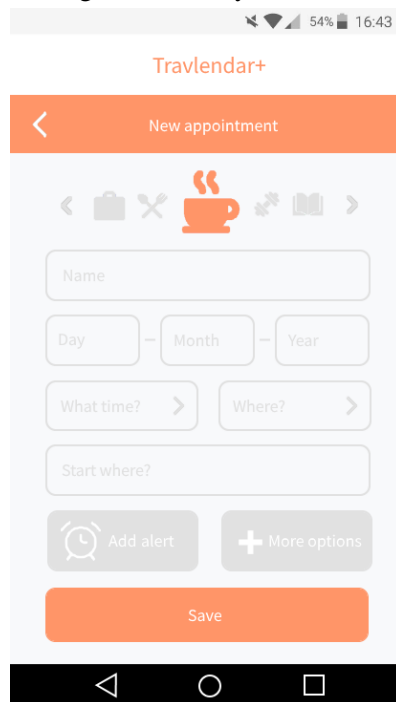
Figure 12: New appointment
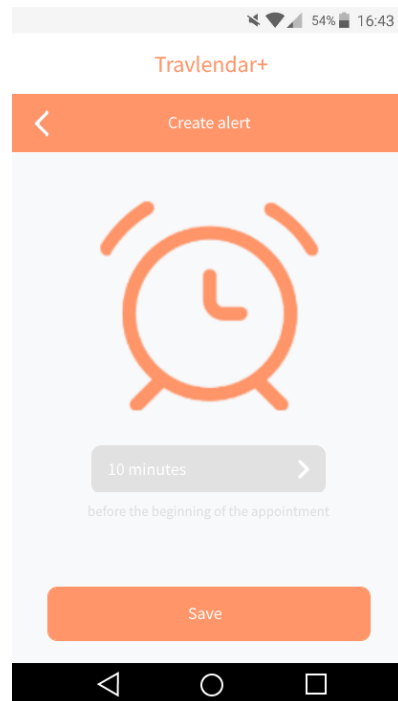
Figure 13: New alert
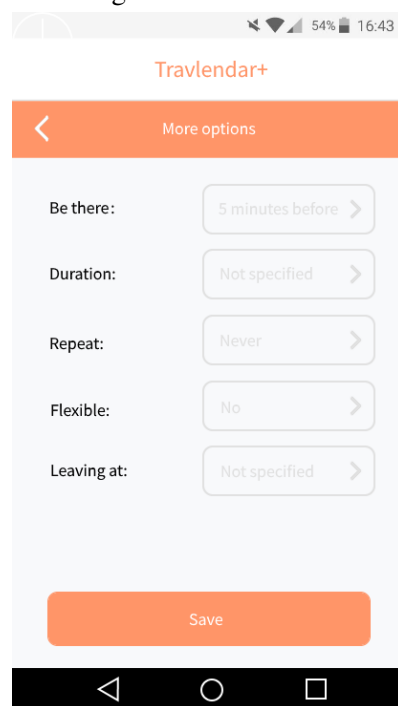
Figure 14: appointment options
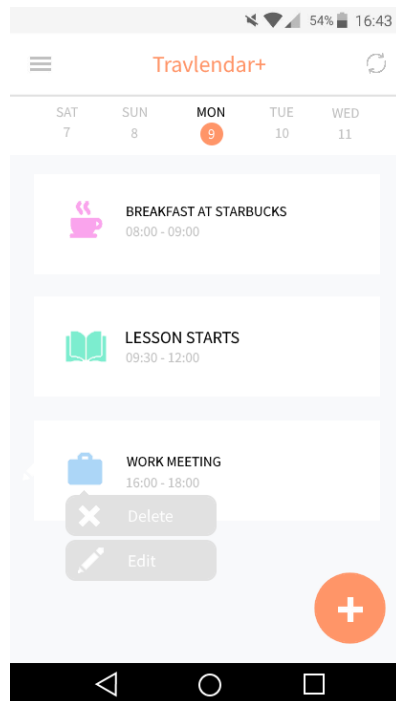
Figure 15: Edit or delete appointment
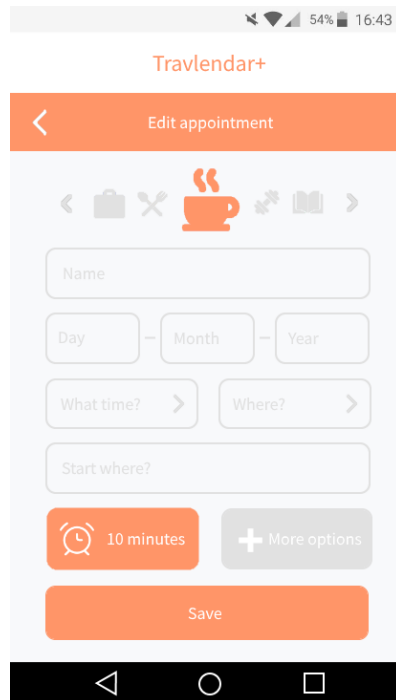
Figure 16: Edit appointment
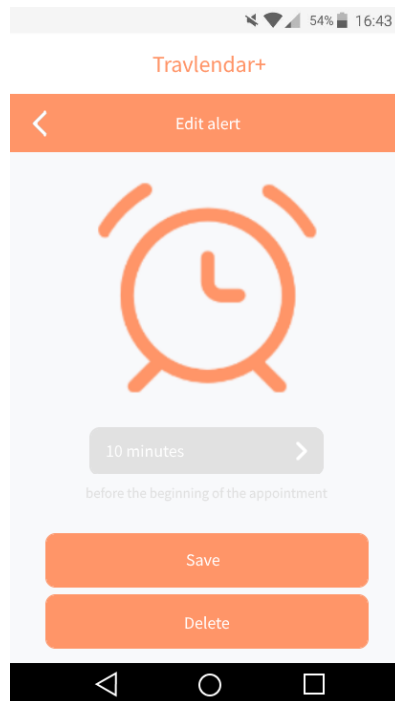
Figure 17: Edit or delete alert
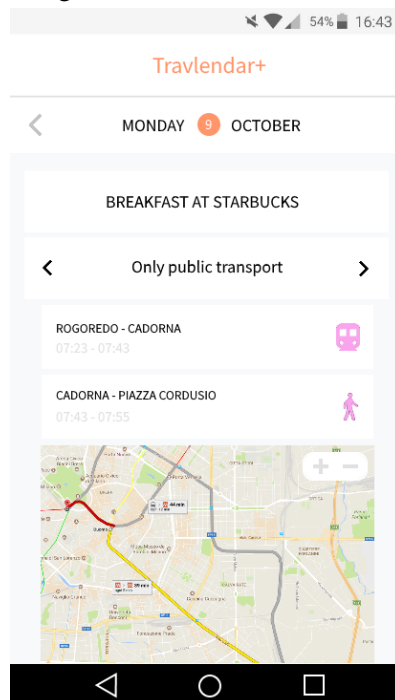
Figure 18: Travel details
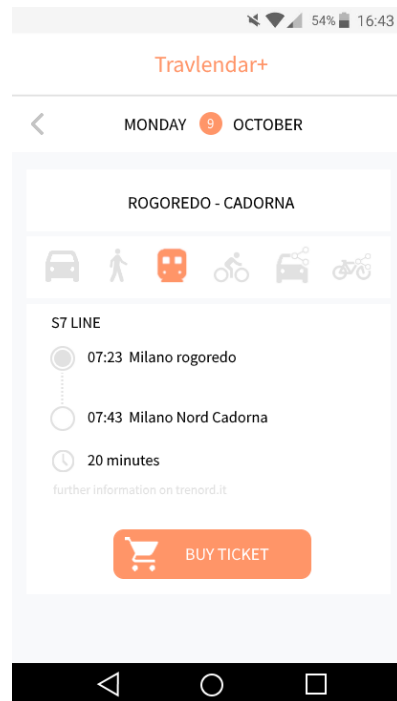
Figure 19: Movement details

3.1.2 Software interface

The application makes uses of the following APIs:

- Weather API: https://openweathermap.org/api

- Google Maps API: https://developers.google.com/maps/

- Trenord API: https://github.com/bluviolin/TrainMonitor/wiki/API-del-sistema-Viaggiatreno

- Car2go API: https://github.com/car2go/openAPI

- Enjoy API: https://github.com/mattiaongit/enjoy/blob/master/enjoy.py

- BikeMi API: https://github.com/pierlauro/bikemi-unofficial-api

- MoBike API: https://github.com/ubahnverleih/WoBike

3.2 Domain assumptions

- **[D1]** The user is always connected via Internet and the connection is stable.

- **[D2]** Information about weather, transport means, maps and travel times are provided by external APIs.

- **[D3]** Data provided by external APIs are correct.

- **[D4]** The process of ticket payment is made through an external public transport service.

- **[D5]** If the payment is fulfilled without errors, the tickets are correctly received.

3.3 Functional requirements

- **R1]** The user must be logged into the system to access application features.

- **[R2]** The user must be able to choose the option of creating a new appointment.

- **[R3]** The user must be able to choose the option of editing a selected appointment.

- **[R4]** The user must be able to choose the option of deleting a selected appointment.

- **[R5]** The system must be able to provide the user with an overview of his calendar and the user must be able to view all appointments fixed in a certain period.

- **[R6]** The user must be able to select a chosen day from the overview of his calendar.

- **[R7]** The user must be able to select a specific appointment in his calendar.

- **[R8]** The system must ask the user to provide all information needed for the creation of a new appointment, such as place and time of start and overall duration.

- **[R9]** The system must check if the information provided by the user are correct.

- **[R10]** The system must check if an appointment overlaps with other events and must eventually notify it to the user.

- **[R11]** The system must give the user access to all details of a selected appointment and the user must be allowed to edit the information needed.

- **[R12]** The user must be able to set advanced information for a created appointment.

- **[R13]** The user must be able to set an appointment as flexible, specifying the interval of time.

- **[R14]** The user must be able to set an appointment as repeatable, specifying the desired days.

- **[R15]** The system must schedule any flexible or repeatable appointment in the correct way, avoiding overlapping with other appointments.

- **[R16]** The appointment intended to be modified must have been previously successfully created and not already deleted.

- **[R17]** The user must confirm the creation of the new appointment.

- **[R18]** The user must confirm any appointment modification.

- **[R19]** The system must save the user modifications in memory and the calendar must be updated.

- **[R20]** The system must remove a deleted appointment from the memory and delete every alert related to it.

- **[R21]** The user must be able to switch between different possible calendar, such as daily calendar, weekly calendar and monthly calendar.

- **[R22]** The system must be able to provide information about the scheduled travels for a chosen day, showing the transport means and the estimated time required from each travel.

- **[R23]** The system must choose the best option between the possible travel alternatives according to the preferences expressed in the user profile settings and the information about external weather.

- **[R24]** The user must be able to select a specific travel in his daily schedule.

- **[R25]** The system must provide detailed information about the travels selected by the user, such as the trace route on the map and the weather conditions.

- **[R26]** The system must provide the user with an overview of the possible travel alternatives for the chosen travel, specifying all details for each one.

- **[R27]** The user must be able to filter the travel alternatives furnished by the system according to defined parameters, such as time of travelling or overall cost.

- **[R28]** The user must be able to choose a favourite travel option different from the displayed default one.

- **[R29]** The user must be able to select a specific movement in a travel.

- **[R30]** The system must provide detailed information about the movements selected by the user, such as the specific trace route on the map and the price of the ticket.

- **[R31]** The user must be able to choose an alternative transport mean for a selected movement, if there are any.

- **[R32]** The system must update the daily schedule according to the travel option chosen by the user and the user must be able to see the new updated schedule.

- **[R33]** The system must give to the user the possibility of buying the ticket for the selected travel.

- **[R34]** The system must save a copy of the bought tickets.

- **[R35]** The user must be able to access to a ticket page from the home page.

- **[R36]** The system must provide a list of all the bought tickets and the user must be able to select and view a specific one in full screen.

- **[R37]** The user must be able to access the preferences panel of his account.

- **[R38]** The system must give the user the possibility of setting various preferences, such as owned and preferred travel means, address of Home and other general travel preferences.

- **[R39]** The user must be able to edit the provided preferences when needed.

- **[R40]** The system must give the user the possibility of adding an alert to an appointment while it is being created or modified.

- **[R41]** The user must be able to choose a desired interval of time for the warning alert.

- **[R42]** The user must confirm the alert creation and the system must save the insertion in the memory.

- **[R43]** The user must be able to modify or remove the inserted alert when needed.

- **R44]** In case of any alert modification made by the user, the user must confirm the modification and the system must save all changes.

3.4   Goals

3.4.1   [G1] Allow an User to create a new appointment in his calendar.

- **[R1]** The user must be logged into the system to access application features.

- **[R5]** The system must be able to provide the user with an overview of his calendar and the user must be able to view all appointments fixed in a certain period.

- **[R6]** The user must be able to select a chosen day from the overview of his calendar.

- **[R2]** The user must be able to choose the option of creating a new appointment.

- **[R8]** The system must ask the user to provide all information needed for the creation of a new appointment, such as place and time of start and overall duration.

- **[R40]** The system must give the user the possibility of adding an alert to an appointment while it is being created or modified.

- **[R9]** The system must check if the information provided by the user are correct.

- **[R12]** The user must be able to set advanced information for a created appointment.

- **[R10]** The system must check if an appointment overlaps with other events and must eventually notify it to the user.

- **[R17]** The user must confirm the creation of the new appointment.

- **[R19]** The system must save the user modifications in memory and the calendar must be updated.

3.4.2   [G2] Allow a user to create flexible and repeatable appointments (such as lunches, study breaks etc.)

- **[R1]** The user must be logged into the system to access application features.

- **[R2]** The user must be able to choose the option of creating a new appointment.

- **[R12]** The user must be able to set advanced information for a created appointment.

- **[R13]** The user must be able to set an appointment as flexible, specifying the interval of time.

- **[R14]** The user must be able to set an appointment as repeatable, specifying the desired days.

- **[R15]** The system must schedule any flexible or repeatable appointment in the correct way, avoiding overlapping with other appointments.

3.4.3 [G3] Allow a user to edit an existing appointment in his calendar.

- **[R1]** The user must be logged into the system to access application features.

- **[R16]** The appointment intended to be modified must have been previously successfully created and not already deleted.

- **[R5]** The system must be able to provide the user with an overview of his calendar and the user must be able to view all appointments fixed in a certain period.

- **[R7]** The user must be able to select a specific appointment in his calendar.

- **[R3]** The user must be able to choose the option of editing a selected appointment.

- **[R11]** The system must give the user access to all details of a selected appointment and the user must be allowed to edit the information needed.

- **[R40]** The system must give the user the possibility of adding an alert to an appointment while it is being created or modified.

- **[R9]** The system must check if the information provided by the user are correct.

- **[R10]** The system must check if an appointment overlaps with other events and must eventually notify it to the user.

- **[R18]** The user must confirm any appointment modification.

- **[R19]** The system must save the user modifications in memory and the calendar must be updated.

3.4.4 [G4] Allow a user to delete an existing appointment from his calendar.

- **[R1]** The user must be logged into the system to access application features.

- **[R5]** The system must be able to provide the user with an overview of his calendar and the user must be able to view all appointments fixed in a certain period.

- **[R16]** The appointment intended to be modified must have been previously successfully created and not already deleted.

- **[R7]** The user must be able to select a specific appointment in his calendar.

- **[R4]** The user must be able to choose the option of deleting a selected appointment.

- **[R18]** The user must confirm any appointment modification.

- **[R20]** The system must remove a deleted appointment from the memory and delete every alert related to it.

- **[R19]** The system must save the user modifications in memory and the calendar must be updated.

3.4.5 [G5] Allow a user to check his calendar to see his appointments.

- **[R1]** The user must be logged into the system to access application features.

- **[R5]** The system must be able to provide the user with an overview of his calendar and the user must be able to view all appointments fixed in a certain period.

- **[R21]** The user must be able to switch between different possible calendar, such as daily calendar, weekly calendar and monthly calendar.

3.4.6 [G6] Allow a user to view his Daily Schedule.

- **[R1]** The user must be logged into the system to access application features.

- **[R6]** The user must be able to select a chosen day from the overview of his calendar.

- **[R22]** The system must be able to provide information about the scheduled travels for a chosen day, showing the transport means and the estimated time required from each travel.

- **[R23]** The system must choose the best option between the possible travel alternatives according to the preferences expressed in the user profile settings and the information about external weather.

- **[D1]** The user is always connected via Internet and the connection is stable.

- **[D2]** Information about weather, transport means, maps and travel times are provided by external APIs.

- **[D3]** Data provided by external APIs are correct.

3.4.7 [G7] Allow a user to see all the details of a specific travel.

- **[R1]** The user must be logged into the system to access application features.

- **[R22]** The system must be able to provide information about the scheduled travels for a chosen day, showing the transport means and the estimated time required from each travel.

- **[R24]** The user must be able to select a specific travel in his daily schedule.

- **[R25]** The system must provide detailed information about the travels selected by the user, such as the trace route on the map and the weather conditions.

- **[R29]** The user must be able to select a specific movement in a travel.

- **[R30]** The system must provide detailed information about the movements selected by the user, such as the specific trace route on the map and the price of the ticket.

- **[D1]** The user is always connected via Internet and the connection is stable.

- **[D2]** Information about weather, transport means, maps and travel times are provided by external APIs.

- **[D3]** Data provided by external APIs are correct.

3.4.8 [G8] Allow a user to navigate and choose between different travel alternatives.

- **[R1]** The user must be logged into the system to access application features.

- **[R24]** The user must be able to select a specific travel in his daily schedule.

- **[R26]** The system must provide the user with an overview of the possible travel alternatives for the chosen travel, specifying all details for each one.

- **[R27]** The user must be able to filter the travel alternatives furnished by the system according to defined parameters, such as time of travelling or overall cost.

- **[R28]** The user must be able to choose a favourite travel option different from the displayed default one.

- **[R29]** The user must be able to select a specific movement in a travel.

- **[R31]** The user must be able to choose an alternative transport mean for a selected movement, if there are any.

- **[R32]** The system must update the daily schedule according to the travel option chosen by the user and the user must be able to see the new updated schedule.

- **[D1]** The user is always connected via Internet and the connection is stable.

- **[D2]** Information about weather, transport means, maps and travel times are provided by external APIs.

- **[D3]** Data provided by external APIs are correct.

3.4.9 [G9] Allow a user to manage alerts for each appointment.

- **[R40]** The system must give the user the possibility of adding an alert to an appointment while it is being created or modified.

- **[R41]** The user must be able to choose a desired interval of time for the warning alert.

- **[R42]** The user must confirm the alert creation and the system must save the insertion in the memory.

- **[R43]** The user must be able to modify or remove the inserted alert when needed.

- **[R44]** In case of any alert modification made by the user, the user must confirm the modification and the system must save all changes.

3.4.10 [G10] Allow a user to manage his travel preferences.

- **[R37]** The user must be able to access the preferences panel of his account from the home page.

- **[R38]** The system must give the user the possibility of setting various preferences, such as owned and preferred travel means, address of Home and other general travel preferences.

- **[R39]** The user must be able to edit the provided preferences when needed.

3.4.11 [G11] Allow a user to buy public transportation tickets.

- **[R1]** The user must be logged into the system to access application features.

- **[R29]** The user must be able to select a specific movement in a travel.

- **[R33]** The system must give to the user the possibility of buying the ticket for the selected travel.

- **[R34]** The system must save a copy of the bought tickets.

- **[D5]** The payment process and ticket acquisition is made by an external public transport service.

- **[D1]** The user is always connected via Internet and the connection is stable.

- **[D4]** The process of ticket payment is made through an external public transport service.

- **[D5]** If the payment is fulfilled without errors, the tickets are correctly received.

3.4.12 [G12] Allow a user to view the previously bought tickets.

- **[R1]** The user must be logged into the system to access application features.

- **[R34]** The system must save a copy of the bought tickets.

- **[R35]** The user must be able to access to a ticket page from the home page.

- **[R36]** The system must provide a list of all the bought tickets and the user must be able to select and view a specific one in full screen.

3.5 Design constraints

3.5.1 Standards compliance

The application must require to the user different permissions:

- Access to the calendar;

- Get hit position with GPS;

- Access to device storage.

3.5.2 Hardware limitations

The application, at the moment, runs only on Android 4.0.3 version or newer.
The device needs:

- Internet connection;

- GPS;

- Space for save application in memory.

Actual devices on the market satisfy all these requirements.

3.6 Software system attributes

3.6.1 Reliability

The system must guarantee a 24/7 service.

3.6.2 Availability

The system requires a GPS service and internet connection in order to work properly. When the connection is down the system works with the last updated information available in the device memory.

3.6.3 Security

The application must provide secure storage for all sensitive data inserted by the user. One way to achieve it is the use of cryptographical techniques.

3.6.4 Maintainability

The application will be released in beta version, meaning that it may eventually have some bugs. The application will be periodically upgraded and each release will be focus on solving detected problems. Periodically all information stored in the application must be backed up, in order to reduce the possibility of losing information in case of malfunctions.

3.6.5   Portability

The application will be released only for Android smartphones (more specifically only for Android version 4.0.3 Ice Cream Sandwich or later).
Future plan is to lead the software also on iOS devices.

4 Scenarios

4.1 Scenario 1

Andrea has just booked a last minute flight from Milan Orio al Serio airport to Prague, in Czech Republic, but being not a frequent flyer, he's pretty worried about the idea of losing his plane. Furthermore, he lives far away from the airport and he needs to reach it by public transportation. Two days before his departure, he decides to download Travlendar+ app. He creates a new account by connetting his Facebook profile and fills out the essential account settings, giving his basic preferences (he doesn't have any car or bike and he prefers cheaper travels). Then he creates a new appointment called "Prague"in his calendar, inserting the date of Saturday 11/11/17, the time 13:00 and the location of the airport. He chooses the option "I want to be there..." and select "2 hours before". Then he creates the new appointment. His calendar now shows the "Prague" event, and the app easily displays how to reach the airport in time, by leaving home at 9:32, walking until the nearest metro station, taking the metro until Stazione Centrale and then taking a public bus to the airport at 10:10. Now Andrea feels much more confident!

4.2 Scenario 2

Serena has been using the Travlendar+ app since a couple of weeks. She has just bought a new car that allows her to move trough the city in a easier way, and wants the app to consider that when it display the optimal travel solution. She opens the app and moves to the preferences panel of her account. She then select "Travel means owned" and puts a tick in the "car" option. Then, she open her calendar and check her daily schedules for the next three days. Some of the travels suggested by the app are now changed and replaced with faster and more confortable movements with car.

4.3 Scenario 3

Marco has a scheduled appointment in program for the following day, and the app used to suggest a 7 minutes-movement by bike until the train station. But now the weather widget seems to announce a rainy day, and the app switched to a warmer travel by metro. But Marco is not afraid of rain and likes walking, so he opens the daily schedule, selects the metro movement and checks the possible travel alternatives. He then chooses the option "by walk". The app nows shows the updated schedule, and the system will remember the choice for the future.

4.4 Scenario 4

Riccardo is a pretty absent-minded man, and has a morning full of appointments in schedule for the next day. He has added all the meetings in his calendar, but must absolutely not forget them, so he decides to add an alarm to each of them, in order to remember his commitments. He opens the app and select one by one all his appointments of Tuesday. For each one, he taps on "add an alert to this appointment" and chooses to be warned 15 minutes before the start.

5 UML modelling

5.1 Use case

Figure 20: Use case diagram

30

5.1.1 Sign up

Name	Sign up
Actors	Guest
Entry conditions	The guest is on the log in page of the application and clicks on "Register" button.

Flow of events	1. A pop-up shows up asking to the guest if he wants to connect an existing account, such as Google or Facebook, or if he want to create a new account.
	
	   (a) If the guest connects his account, the system accepts the request and creates a new Travlendar+ account based on provided account.
	
	   (b) If the guest chooses to create a new account, he is redirected to the registration page which contains all the fields to be filled.
	
	 2. The guest fills out all the mandatory fields.
	
	 3. (Optional) The guest adjusts the preferences settings.
	
	 4. The guest clicks on button "Confirm".
	
	 5. The system checks data provided and eventually creates and registers the user account.

Exit conditions	The guest has successfully created a new account and he can log into the system with his credentials.
Exceptions	• Email provided is already in use. The system does not proceed in the registration process and the account is not created. It is possible to repeat the procedure. • Data provided are incorrect. The system highlights the incorrect fields and asks the user to repeat the procedure.

     31

5.1.2 Log in

Name	Log in
Actors	User
Entry conditions	The user launches the application and clicks on the "Log In" button.
Flow of events	1. The user inserts his email. 2. The user inserts his password. 3. The user clicks on the "Log In" button.
Exit conditions	The login procedure is successfully completed. The user is logged into the system and is able to access to all the functionalities.
Exceptions	The credentials provided are not associated to any existing account. The login procedure is rejected and the guest is brought back to the login page. It is possible to repeat the procedure.

5.1.3 Manage preferences

Name	Manage preferences
Actors	User
Entry conditions	The user clicks on "Preferences" from the side menu on the homepage.
Flow of events	1. The system shows the preferences settings, that include transport means owned, favorite kind of travel and other advanced options. 2. The user accesses to the desired settings and adjusts the preferences as wanted. 3. The user saves the changes.
Exit conditions	The changes are saved and remembered from the system. The preferences have been updated.
Exceptions	The user clicks on "back" without having confirmed the creation. The new preferences are not saved and the application returns to the homepage. It is possible to repeat the procedure.

5.1.4 View daily schedule

Name	View daily schedule
Actors	User, External APIs
Entry conditions	The user clicks on the "View daily schedule" button while checking an appointment on his calendar.
Flow of events	1. The system receives data from different external APIs about routes, traffic, weather and available transport means and computes the best travel option according to the user preferences. 2. The system provides a graphic overview of all the travels scheduled for the day, with the relative movements. 3. The user checks all the information needed and eventually clicks on a specific travel or movement to get further information.
Exit conditions	User has obtained all the information needed and has clicked "back" to return to the homepage or has selected a specific travel/movement to get further information.
Exceptions	• Some of the appointments overlap. The system is not able to compute travels between the appointments and forces the user to choose a prior appointment. • Some of the appointments are not reachable in the allotted time. The system doesn't show the travel and signals the problem to the user with a warning.

5.1.5 View travel details

Name	View travel details
Actors	User, External APIs
Entry conditions	The user is checking his daily schedule and clicks on a specific travel.
Flow of events	1. The system receives data from different external APIs and collects detailed information about the travel, such as the specific itinerary on the map and the weather conditions. 2. The user is redirected to a new page that displays all the information about the selected travel. 3. The user checks all the information needed and eventually clicks on a specific movement to get further information or buy a ticket.
Exit conditions	User has obtained all the information needed and has clicked "back" to return to the daily schedule or has selected a specific movement to get further information.
Exceptions	No exceptions expected

5.1.6 Edit travel

Name	Edit travel
Actors	User
Entry conditions	The user is checking a specific travel.

Flow of events	1. The user provides modifications to the travel as followed:
	
	  (a) Clicking on the travel preferences and switching to a different alternative.
	  (b) Clicking on a specific movement and changing the transport mean.
	
	2. The user saves the changes.

Exit conditions	The changes are saved and the system displays the new modified travel.
Exceptions	There are no possible alternatives for the selected travel. The system displays a warning to notify the user.

5.1.7 View specific movement details

Name	View specific movement details
Actors	User
Entry conditions	User is checking a specific travel and select a specific movement.
Flow of events	1. The system receives data from different external APIs and collects detailed information about the movement, such as the specific itinerary on the map, the weather conditions and the eventual availability of tickets. 2. The user is redirected to a new page that displays all the information about the selected movement. 3. The user checks all the information needed and eventually proceeds in buying a ticket.
Exit conditions	User has obtained all the information needed and has clicked "back" to return to the travel page or has proceeded in buying a ticket.
Exceptions	No exceptions expected

5.1.8 Choose prior appointment

Name	Choose prior appointment
Actors	User
Entry conditions	The user is checking the schedule and two appointments overlap.
Flow of events	1. The system signals the user the exactly time overlapping of the appointments. 2. The user clicks on the preferred appointment. 3. The system re-computes the daily schedule giving priority to the chosen appointment.
Exit conditions	No more appointments overlap. The system shows the daily schedule without errors.
Exceptions	No exceptions expected

5.1.9 Buy ticket

Name	Buy ticket
Actors	User, Ticket provider
Entry conditions	User is checking a specific movement and clicks on "Buy ticket"
Flow of events	1. The system connects trough APIs to the external system of the ticket provider. 2. The system shows the user a form to fill with all the payment information. 3. The user fills out all the fields and confirm the payment. 4. The system sends information to the ticket provider and waits for confirmation and ticket data.
Exit conditions	Payment is successfully fulfilled and bought tickets are available for the view. The user is brought back to the movement page.
Exceptions	Payment is rejected. The system notifies the user with a warning. It is possible to repeat the procedure.

5.1.10 Delete appointment

Name	Delete appointment
Actors	User
Entry conditions	The user selects a specific appointment in his calendar (through daily or weekly view) and clicks on "Delete" button.
Flow of events	1. A pop-up shows up, asking the user to confirm the deletion. 2. The user confirms the deletion. 3. The system removes all the appointment information from the memory, alert included.
Exit conditions	The appointment has been deleted and removed from the system. The user is redirected to his calendar page, which has been updated with the removal of the appointment.
Exceptions	The user does not confirm the deletion. The appointment has not been deleted and the user is redirected to his calendar page. It is possible to repeat the procedure.

5.1.11 Create appointment

Name	Create appointment
Actors	User
Entry conditions	The user is checking his calendar and clicks on "Create appointment" button.

Flow of events	1. The user is redirected to the appointment creation page, which contains all the fields required to perform the creation.
	
	2. The user fills out all mandatory fields. The following steps aren't mandatory:
	
	    (a) User chooses among the available icons.
	    (b) User adds an alert to remember the appointment.
	    (c) User clicks on "More options" and provides more detailed options.
	
	3. User confirms the creation.
	
	4. The system saves the appointment information.

Exit conditions	The appointment is created and inserted in the system. The user is redirected to his calendar page, which has been updated with the new appointment.
Exceptions	• The user has provided incorrect information: the appointment is not created and the user must repeat the procedure. • The user clicks on "back" without having confirmed the creation. The appointment is not created and the application returns to the calendar page. It is possible to repeat the procedure. • The appointment overlaps with other previously created appointments. The appointment is created and inserted anyway, but the user receives a notification of the overlapping.

5.1.12 Edit appointment

Name	Edit appointment
Actors	User
Entry conditions	The user selects a specific appointment in his calendar (through daily or weekly view) and clicks on "Edit" button.

Flow of events	1. The user is redirected to the appointment editing page which contains all the previously inserted information.
	
	2. The user can perform the following actions:
	
	    (a) Changing the appointment icon.
	    (b) Editing the information.
	    (c) Adding an alert to the appointment.
	    (d) Clicking on "More options" and providing more detailed options.
	
	3. The user saves the changes.
	
	4. The system saves the updated appointment information.

Exit conditions	The changes are saved and the appointment has been modified. The user is redirected to his calendar page, which has been updated with the new information provided.
Exceptions	• The user has provided incorrect information: the changes are not saved and the user must repeat the procedure. • The user clicks on "back" without having saved the changes. The appointment has not been modified and the application returns to the calendar page. It is possible to repeat the procedure. • The appointment now overlaps with other previously created appointments. The changes as saved anyway, but the user receives a notification of the overlapping.

5.1.13 Create flexible appointment

Name	Create flexible appointment
Actors	User
Entry conditions	The user is creating/editing an appointment, clicks on "More options" and clicks on the "Flexible" field.
Flow of events	1. A pop-up shows up, containing the fields to be filled. 2. The user fills out the fields, specifying the time range of the appointment. 3. The user confirms the choice.
Exit conditions	The choice is saved for later and the user is back on the More option page. At the end of the creation process, the appointment will be created at a time compatible with the time interval provided.
Exceptions	The user clicks on "back" without having confirmed the choice. The appointment has not been modified and the application returns to the More options page. It is possible to repeat the procedure.

5.1.14 Create repeatable appointment

Name	Create repeatable appointment
Actors	User
Entry conditions	The user is creating/editing an appointment, clicks on "More options" and clicks on the "Flexible" field.
Flow of events	1. A pop-up shows up, containing the fields to be filled. 2. The user fills out the fields, specifying the days in which the appointment is wanted to be created. 3. The user confirms the choice.
Exit conditions	The choice is saved for later and the user is back on the More option page. The appointment is now a repeatable appointment.
Exceptions	The user clicks on "back" without having confirmed the choice. The appointment has not been modified and the application returns to the More options page. It is possible to repeat the procedure.

5.1.15 Create alert

Name	Create alert
Actors	User
Entry conditions	The user is on the appointment creation/editing page and clicks on the "Add alert" button.
Flow of events	1. The user is redirected to the alert creation/editing page. 2. The user fills the field specifying the time of the alert. 3. The user confirms the creation. 4. The system saves the alert information.
Exit conditions	The alert has been created and inserted in the system. The application returns to the appointment creation/editing page and it is now possible to edit the alert.
Exceptions	The user clicks on "back" without having confirmed the creation. The alert is not created and the application returns to the event creation page. It is possible to repeat the procedure.

41

5.1.16 Edit alert

Name	Edit alert
Actors	User
Entry conditions	The user is on the appointment creation/editing page and clicks on the "Edit alert" button.
Flow of events	1. The user is redirected to the alert creation/editing page. 2. The user replaces needed information with updated ones. 3. The user saves the changes. 4. The system saves the alert information.
Exit conditions	All the changes have been saved and inserted in the system. The application returns to the appointment creation/editing page and it is possible to repeat the procedure.
Exceptions	The user clicks on "back" without having saved the changes. The changes have not been saved and the application returns to the appointment creation/modification page. It is possible to repeat the procedure.

5.1.17 Delete alert

Name	Delete alert
Actors	User
Entry conditions	The user is on the appointment creation/editing page and clicks on the "Edit alert" button.
Flow of events	1. The user is redirected to the alert creation/editing page. 2. The user clicks on "Delete". 3. The user confirms the deletion. 4. The system removes all the alert information from the memory.
Exit conditions	The alert has been deleted and removed from the system. The application returns to the appointment creation/editing page and it is now possible to add a new alert.
Exceptions	The user does not confirm the deletion. The alert has not been deleted and the application returns to the alert editing page. It is possible to repeat the procedure.

5.1.18 Check appointments on calendar

| Name	Check appointments on calendar
Actors	User
Entry conditions	The user is logged in and he is on the homepage of the application.
Flow of events	1. The system shows an overview of the existing appointments. 2. The user moves between the appointments and collects the needed infomation.
Exit conditions	The user has collected the needed information and moves to a different page.
Exceptions	No exception expected.

5.1.19   Change calendar view

Name	Change calendar view
Actors	User
Entry conditions	The user is on the homepage of the application and he is checking his appointments.
Flow of events	1. The user opens the lateral menu. 2. The user clicks on one of available view (daily, weekly or monthly). 3. The system changes the layout according to the user's choice.
Exit conditions	The layout is changed and the user is back on the calendar view.
Exceptions	No exception expected.

5.1.20   View bought tickets

| Name	View bought tickets
Actors	User
Entry conditions	The user is logged into the system and clicks on "My tickets" button on the side menu.
Flow of events	1. The system loads the tickets saved in memory and displays the list to the user. 2. The user chooses a specific ticket and clicks on it. 3. The system displays the full screen ticket.
Exit conditions	The user has checked the needed tickets and clicks "back" to return to the homepage.
Exceptions	No tickets saved in memory. The system notifies the absence of saved tickets to the user.

5.2   Sequence diagram
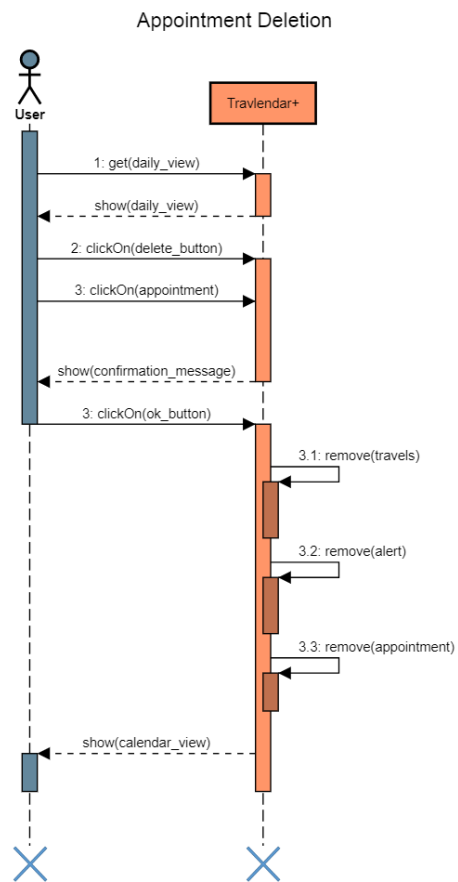
5.2.1   Guest registration
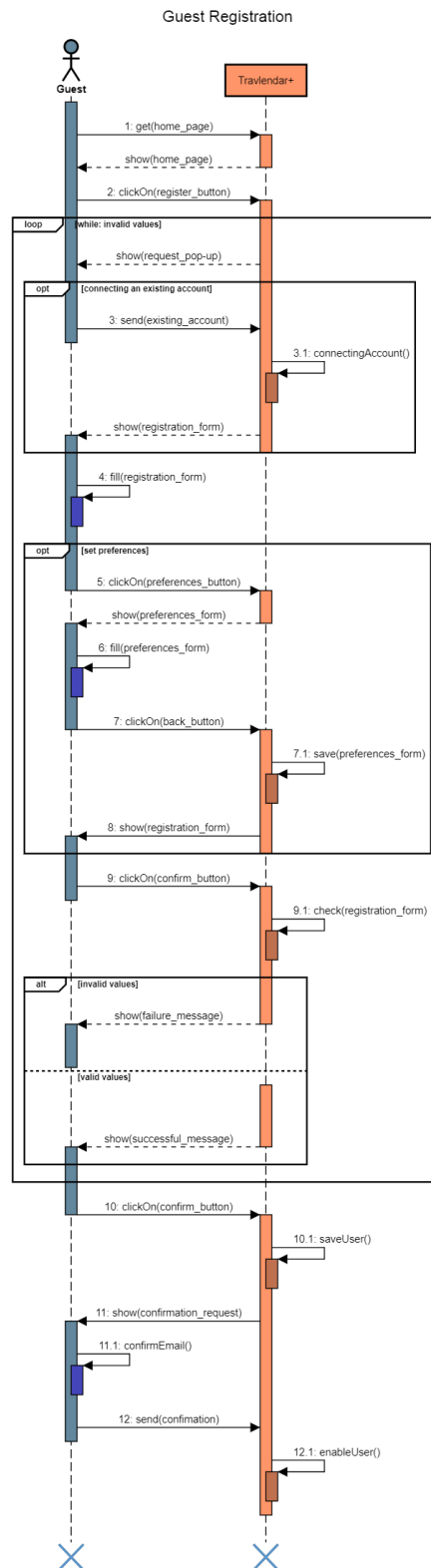
5.2.2 User Login
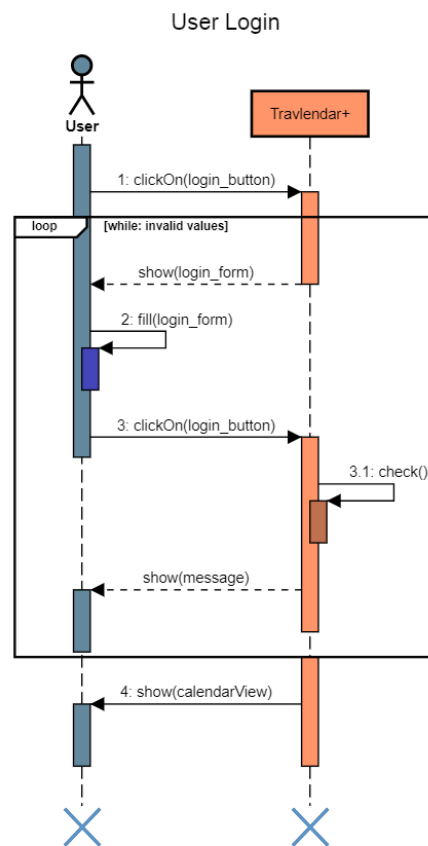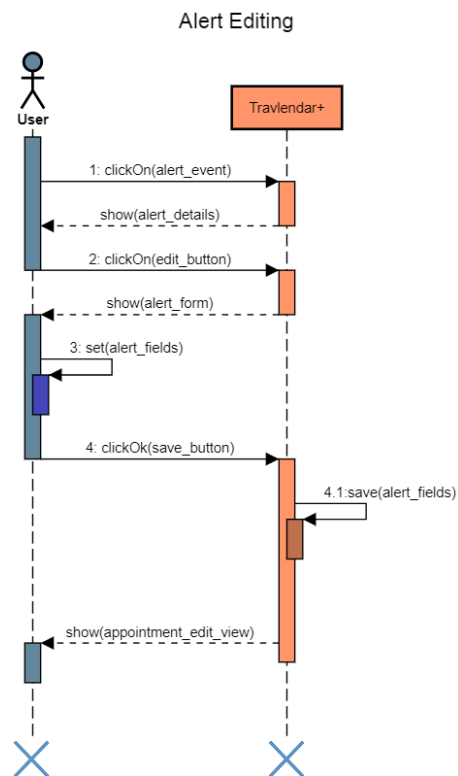
5.2.3 Appointment creation

Appointment Creation
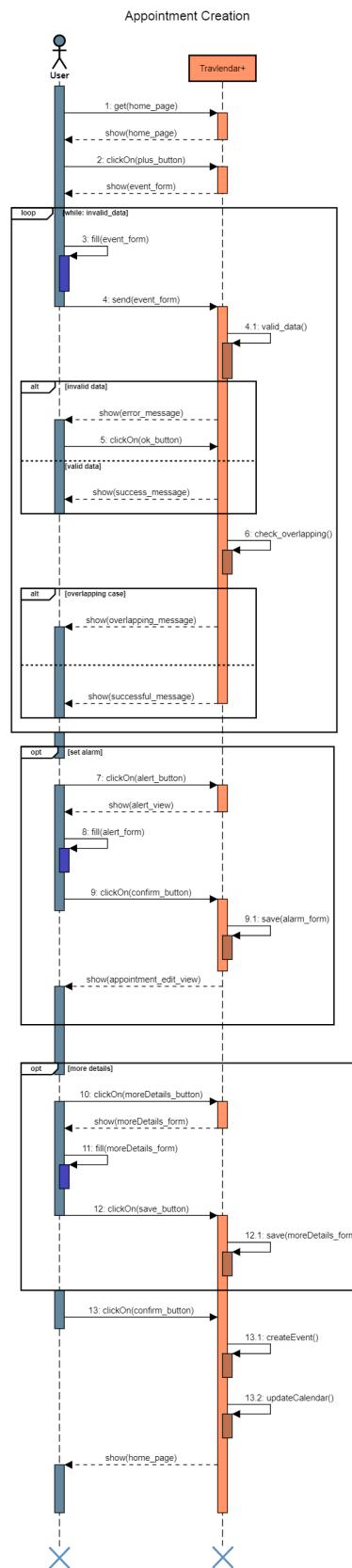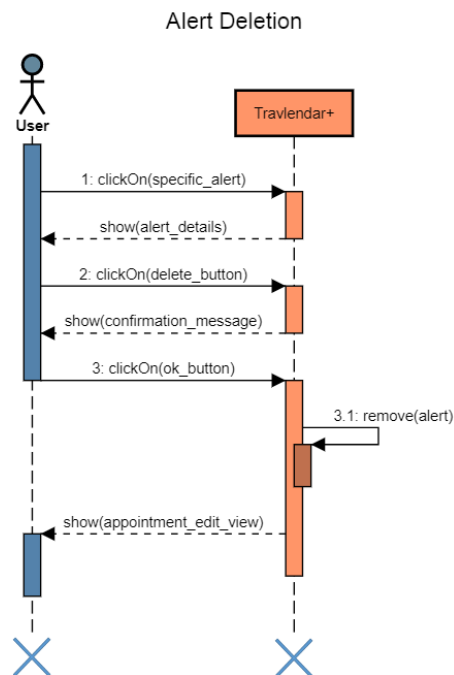
5.2.4 Appointment editing

Appointement Editing
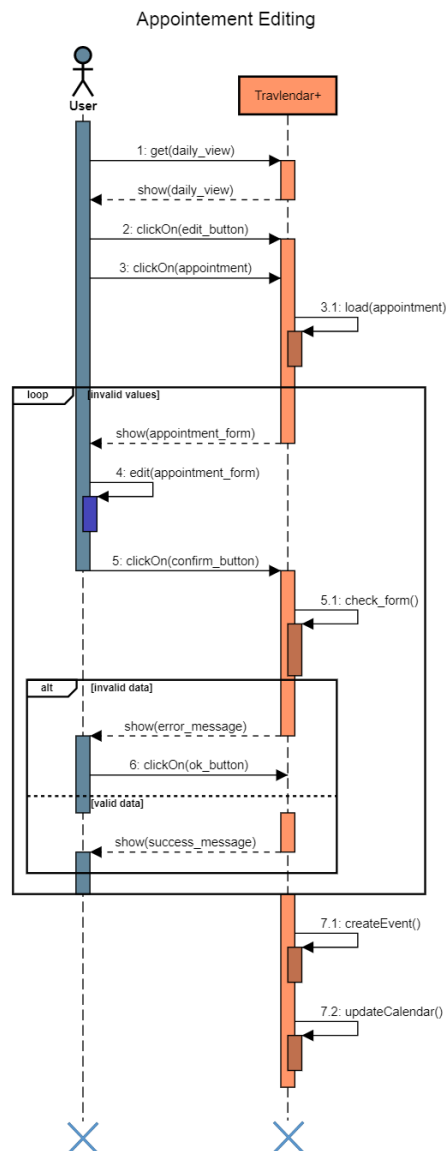
5.2.5 Appointment deletion

5.2.6   Alert editing

Alert Editing

5.2.7 Alert deletion

5.2.8 Scheduling and view travels

Scheduling and view travels

5.2.9 View and edit movements
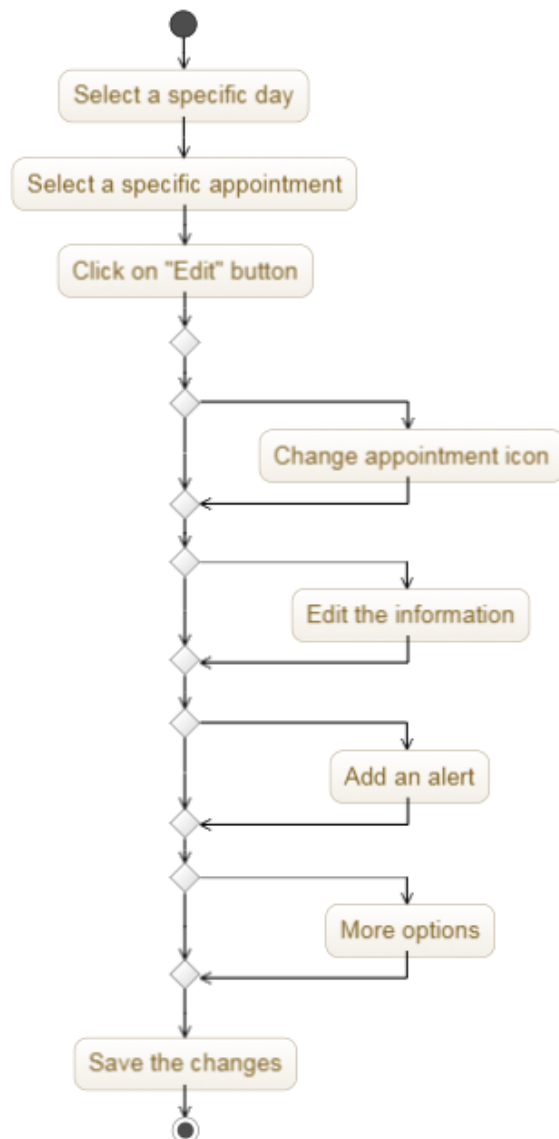

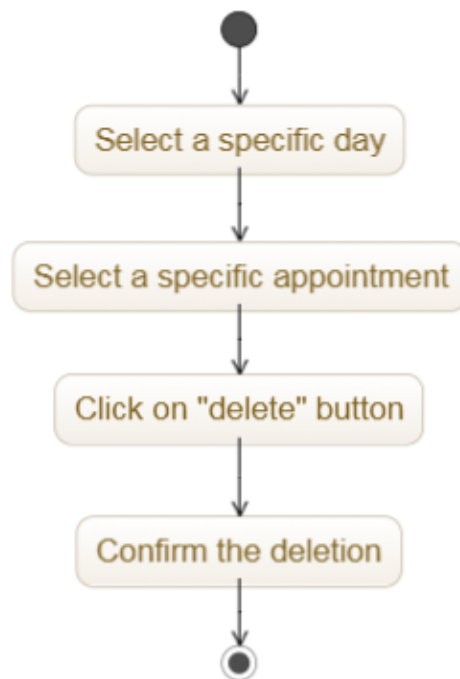View and edit movements
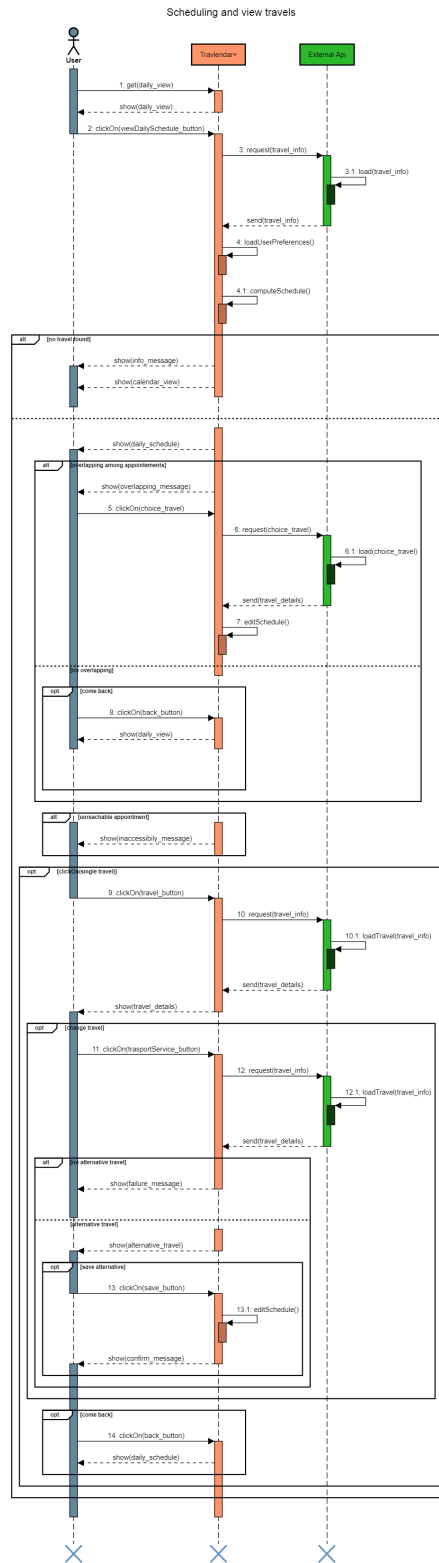
5.3 Activity diagram

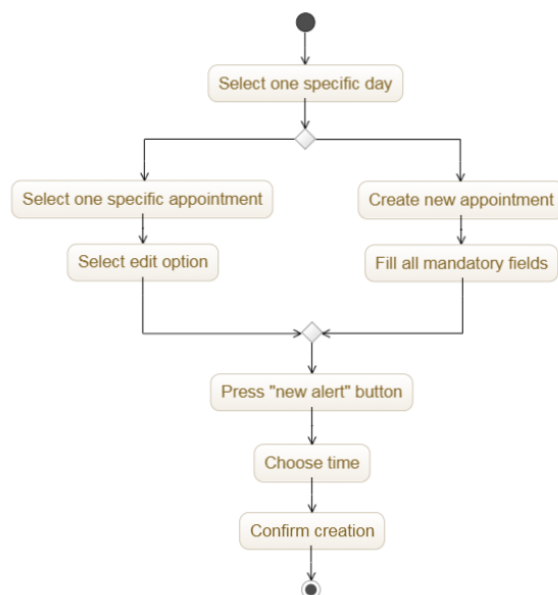### 5.3.1 Create appointment
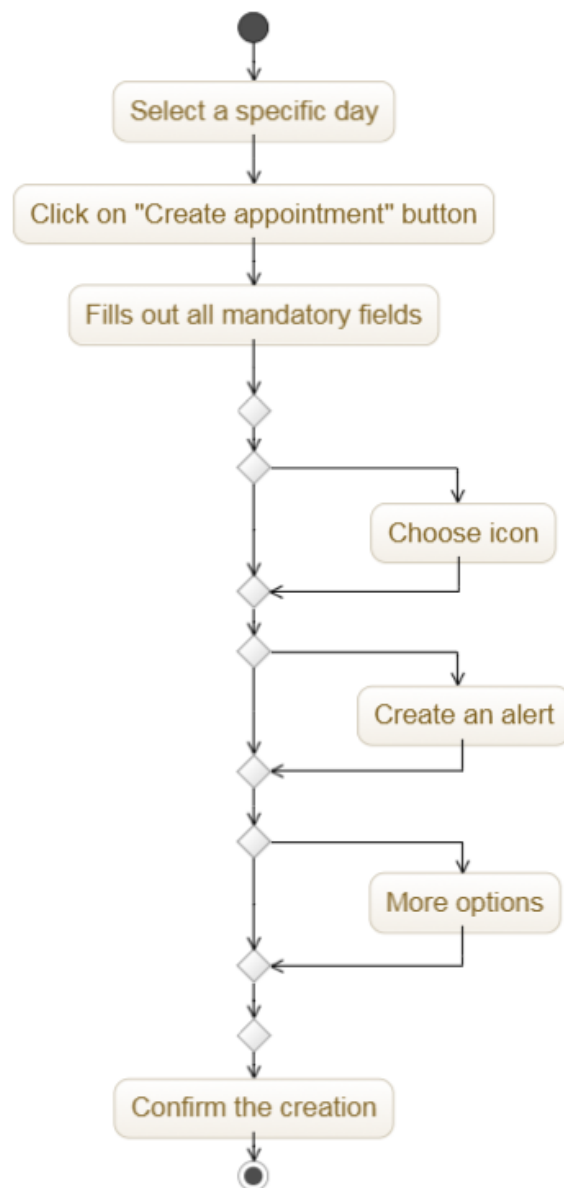
5.3.2 Edit appointment

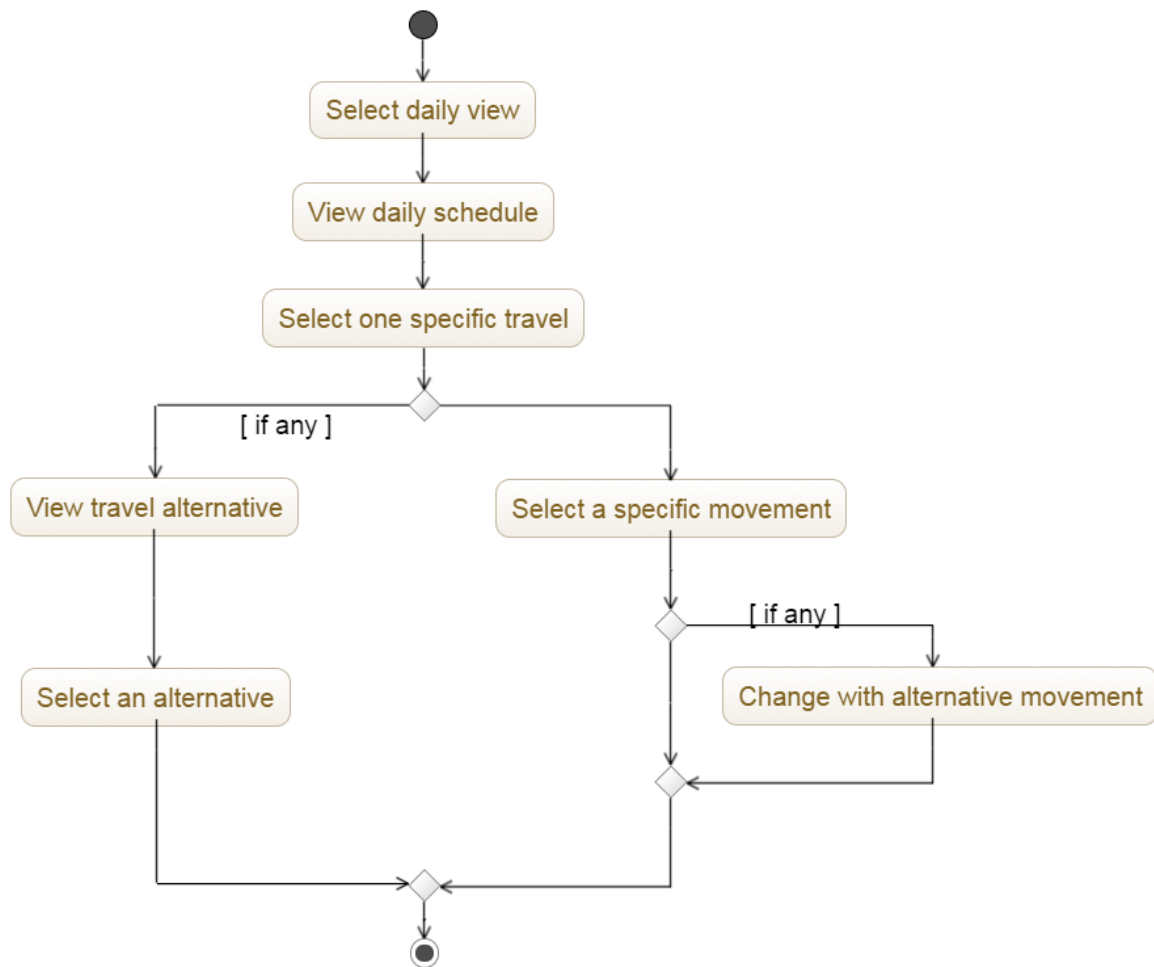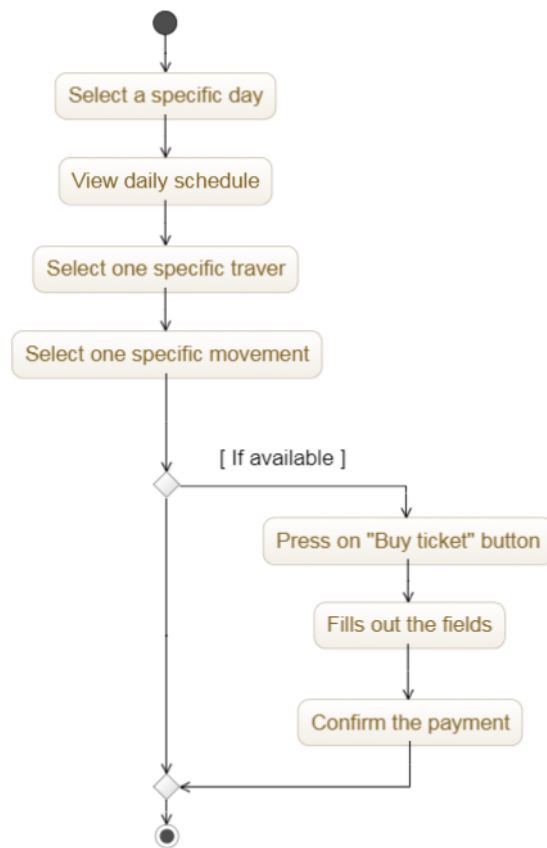### 5.3.3   Delete appointment

5.3.4   Create alert

5.3.5 Edit travel

5.3.6 Buy ticket

6 Formal Analysis Using Alloy

```
open util/integer

//————————MODEL:————————//

sig Name {}

sig Place {}

sig Appointment {
place: one Place,
date: one Int,
time: Int,
duration: lone Int,
alert: lone Alert,
travel: one Travel
} {
date >0
time >0
duration >0
alert != none => alert.appointment = this
travel.placeOfArrival = place
}

sig Alert {
appointment: one Appointment,
date: one Int,
time: one Int
} {
date >0
time >0
appointment.alert = this
}

sig Calendar {
appointments: set Appointment
}

sig User {
calendar: one Calendar,
email: one Name,
password: one Name
}

sig Movement {
placeOfDeparture: one Place,
placeOfArrival: one Place,
extimatedTime: one Int,
} {
```

```
placeOfDeparture != placeOfArrival
extimatedTime > 0
}

sig Travel {
placeOfDeparture: one Place,
placeOfArrival: one Place,
timeOfDeparture: one Int,
extimatedTime: one Int,
movements: some Movement,
} {
placeOfDeparture != placeOfArrival
extimatedTime > 0
timeOfDeparture > 0
}
```

//−−−−−−−−−FACTS:−−−−−−−−//

```
// different users   have different email addresses
fact mailUnique {
no disjoint u1, u2: User | u1.email = u2.email
}

// different users have different calendars
fact calendarUnique {
no disjoint u1, u2: User | u1.calendar = u2.calendar
}

// different calendars have different appointments
fact appointmentsUnique {
all disjoint c1, c2 : Calendar | c1.appointments & c2.appointments = none
}

// a movement cannot exist outside of a travel
fact noMovementWithoutTravel {
Travel.movements = Movement
}

// a travel cannot exist without its appointment
fact noTravelWithoutAppointment {
Appointment.travel = Travel
}

// an appointment cannot exist outside of a calendar
fact noAppointmentWithoutCalendar {
Calendar.appointments = Appointment
}

// a calendar cannot exist without its user
```

```
fact noCalendarWithoutUser {
User.calendar = Calendar
}


// the alert of an appointment cannot be scheduled after the beginning
// of the appointment
fact alertBeforeAppointment {
all a: Appointment | a.alert.date < a.date or (a.alert.date=a.date and
        a.alert.time < a.time)
}


// every travel is composed by a sequence of connected movements
fact travelIsMadeByMovements {

// every travel starts with a movement
all t: Travel | (one m: t.movements | m.placeOfDeparture = t.placeOfDeparture)
// every travel ends with a movement
all t: Travel | (one m: t.movements | m.placeOfArrival = t.placeOfArrival)
// the ending of a movement is the beginning of a new one or
// the end of the travel
all t: Travel | (all m: t.movements | m.placeOfArrival = t.placeOfArrival
or
(one m1: t.movements | m1!=m and m.placeOfArrival = m1.placeOfDeparture))
// the beginning of a movement is the ending of an old one or the
// beginning of the travel
all t: Travel | (all m: t.movements | m.placeOfDeparture = t.placeOfDeparture
or
(one m1: t.movements | m1!=m and m.placeOfDeparture = m1.placeOfArrival ))
// different movements cannot start or end at the same position
all t: Travel | (no disjoint m1, m2: t.movements | m1.placeOfDeparture =
m2.placeOfDeparture
or
m1.placeOfArrival = m2.placeOfArrival )
// no close path
all t: Travel | (no m: t.movements | m.placeOfArrival = t.placeOfDeparture)

}


// the extimated time of a travel is the sum of the extimated times
// of its movements
fact travelTimeSumOfMovementsTime {
all t: Travel | t.extimatedTime = sum (t.movements.extimatedTime)
}


// every traveal leads to the appointment in time
fact travelsInTime {
all a: Appointment | plus[a.travel.timeOfDeparture , a.travel.extimatedTime]
= a.time
}
```

//−−−−−−−ASSERTIONS:−−−−−−//

// if a movement starts at the travel starting and ends at travel ending,
// it's the only movement of the travel (and vice versa)
assert oneMovementTravel {
all t: Travel | all m : t.movements | m.placeOfDeparture =
        t.placeOfDeparture and m.placeOfArrival = t.placeOfArrival <=>
        t.movements = m
}

// a movement nvere takes a longer time than the corresponding travel
assert noMovementLongerThanTravel {
all t:Travel | no m:t.movements | m.extimatedTime > t.extimatedTime
}

//−−−−−−−PREDICATES:−−−−−−//
pred addAppointment [c1, c2: Calendar, a: Appointment] {
a not in c1.appointments implies c2.appointments = c1.appointments + a
}

pred deleteAppointment [c1, c2: Calendar, a: Appointment] {
c2.appointments = c1.appointments − a
}

pred addAlert [a1, a2: Appointment, al: Alert] {
a1.alert = none implies a2.alert = al
}

pred deleteAlert [a1, a2: Appointment, al: Alert] {
a1.alert = al implies a2.alert = none
}

pred show {}

check oneMovementTravel
check noMovementLongerThanTravel

run addAppointment
run deleteAppointment
run addAlert
run deleteAlert

run show for 3 but 8 Int, exactly 1 User, exactly 2 Travel, exactly
3 Movement, exactly 1 Alert
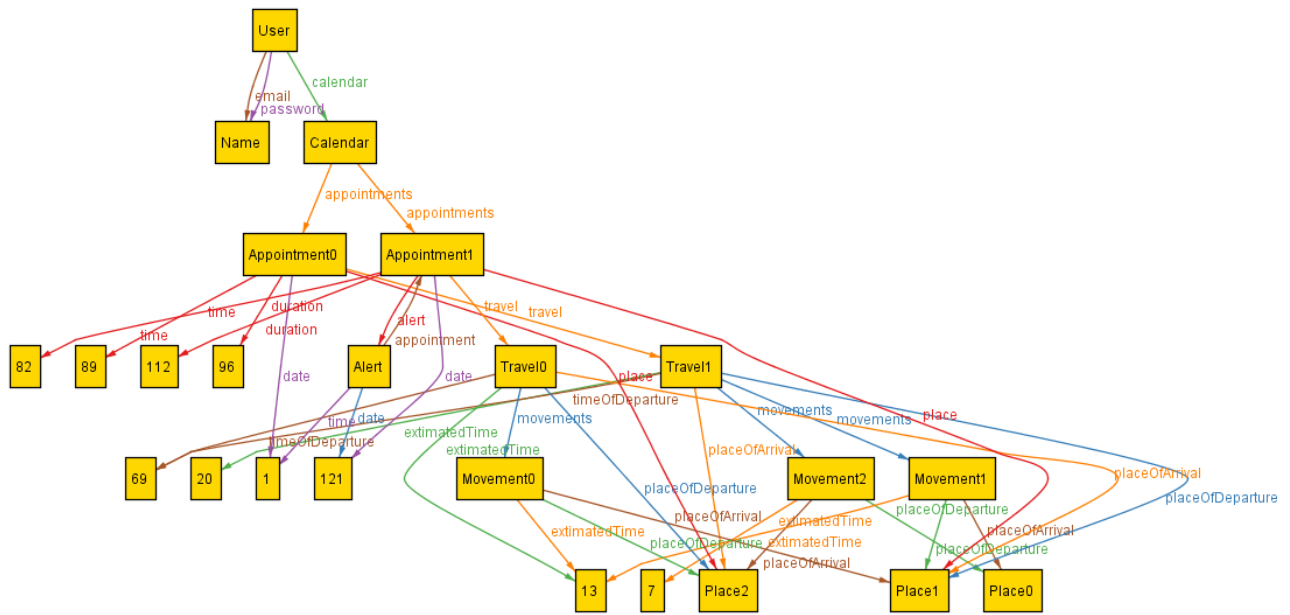run show for 3 but 8 Int, exactly 3 User, exactly 2 Appointment

Figure 21: Alloy: first example

The first picture shows a world in which one user has more appointments in his calendar. Each appointment is associated with a travel:

- Travel0: from Place2 to Place1 (location of Appointment1), extimatedTime = 13;

- Travel1: from Place1 to Place2 (location of Appointment0), extimatedTime = 20;

We notice that:

- Travel0 = Movement0 (Place2 -> Place1, extimatedTime = 13);

- Travel1 = Movement1 (Place1 -> Place0, extimatedTime = 13) + Movement2 (Place0 -> Place2, extimatedTime = 7);
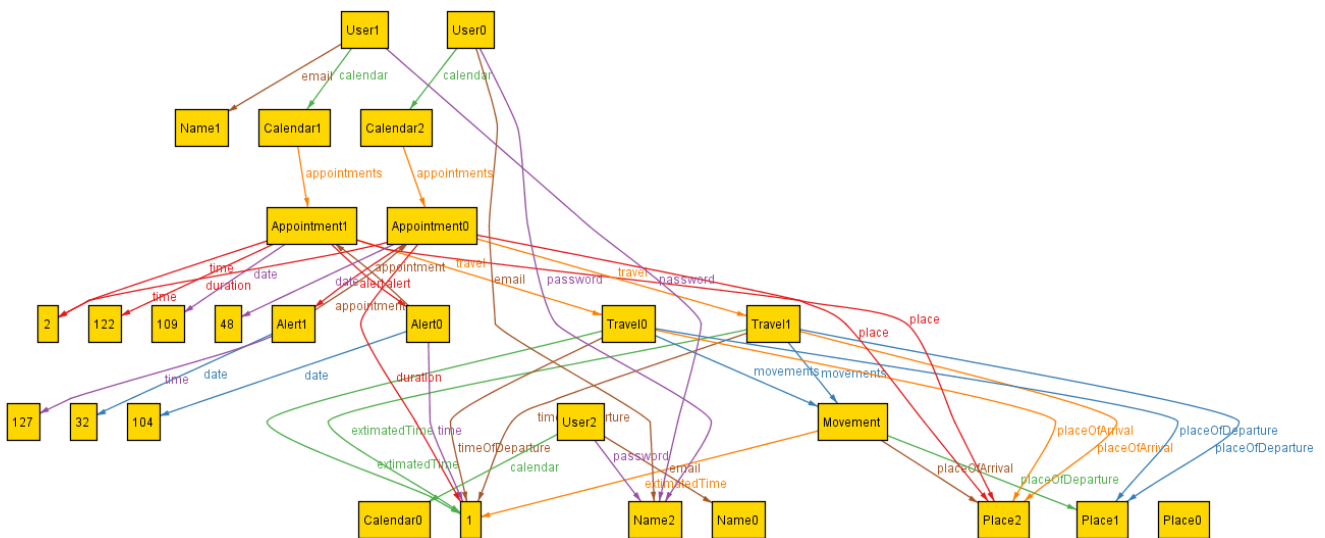

Figure 22: Alloy: second example

The second picture shows a world with more users.

7 Effort Spent

7.1 Hours of work

7.1.1 Andrea Mafessoni

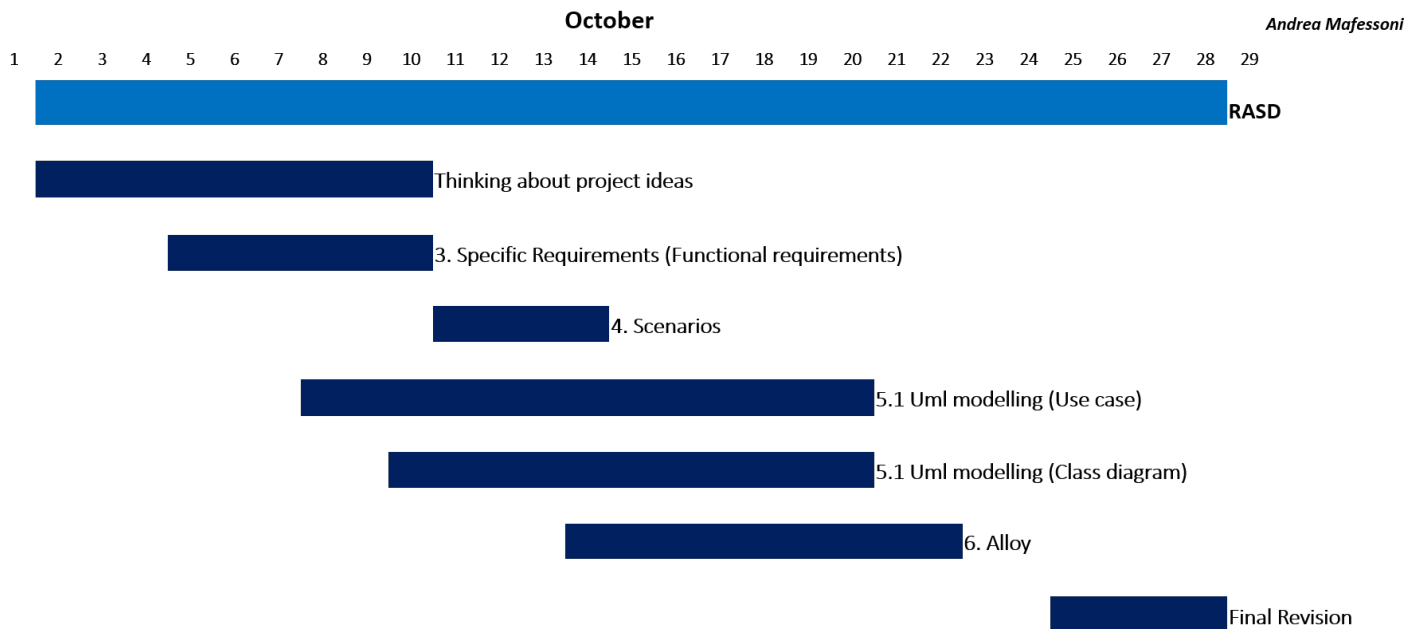| Task name	Start date	End date	Hours(h)	Day
RASD	**02/10/2017**	**28/10/2017**	**34.5**	
Thinking about project ideas	02/10/2017	10/10/2017	5	9
3. Specific Requirements (Functional requirements)	05/10/2017	10/10/2017	5	6
4. Scenarios	11/10/2017	14/10/2017	3.5	4
5.1 Uml modelling (Use case)	08/10/2017	20/10/2017	4	13
5.1 Uml modelling (Class diagram)	10/10/2017	20/10/2017	5	11
6. Alloy	14/10/2017	22/10/2017	7	9
Final revision	25/10/2017	28/10/2017	5	9

Figure 23: Gantt diagram Mafessoni

7.1.2 Andrea Mazzeo

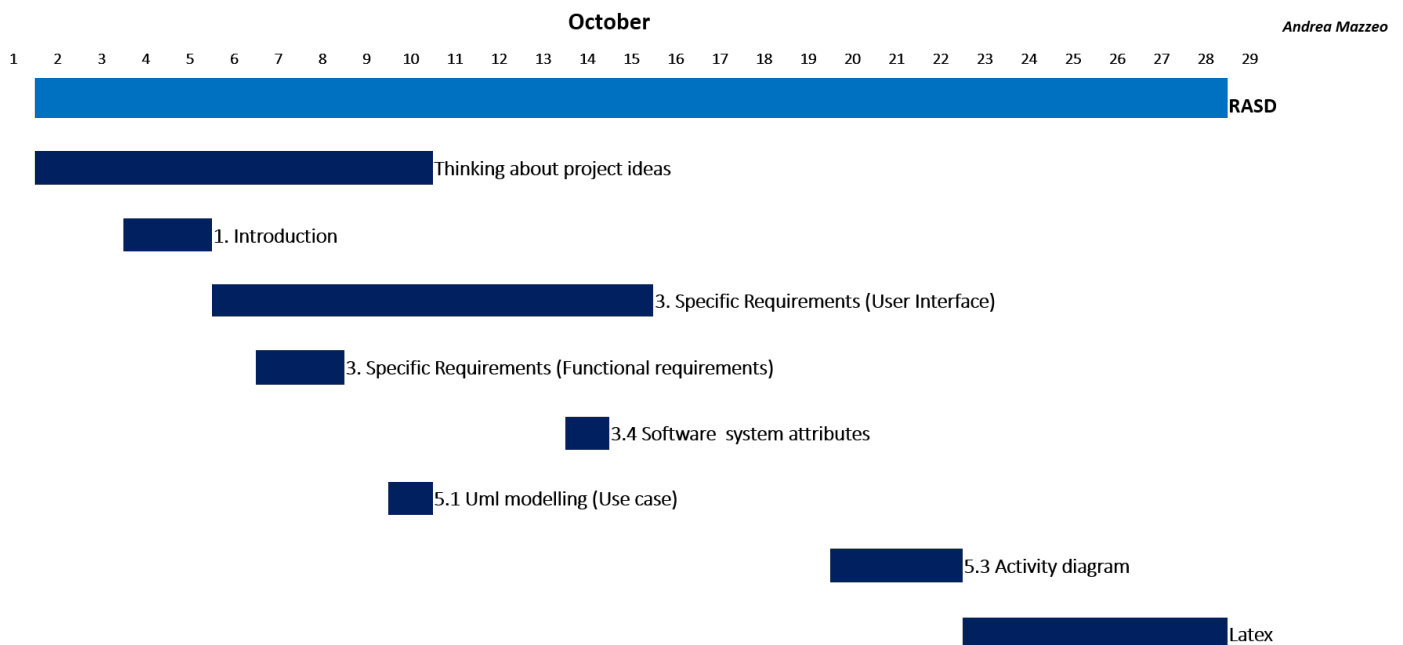| Task name	Start date	End date	Hours(h)	Day
RASD	**02/10/2017**	**28/10/2017**	**35**	
Thinking about project ideas	02/10/2017	10/10/2017	5	9
1. Introduction	04/10/2017	5/10/2017	2.5	1
3. Specific Requirements (User Interface)	06/10/2017	15/10/2017	10	10
3. Specific Requirements (Functional requirements)	07/10/2017	08/10/2017	2.5	2
3.4 Software system attributes	14/10/2017	14/10/2017	1	1
5.3 Activity diagram	20/10/2017	22/10/2017	2	3
5.1 Uml modelling (Use case)	10/10/2017	10/10/2017	2	1
Latex	23/10/2017	28/10/2017	5	6
Final revision	25/10/2017	28/10/2017	5	9

Figure 24: Gantt diagram Mazzeo

7.1.3 Daniele Moltisanti

| Task name	Start date	End date	Hours(h)	Day
RASD	**02/10/2017**	**28/10/2017**	**33.5**	
Thinking about project ideas	02/10/2017	10/10/2017	5	9
2. Overall Description	04/10/2017	06/10/2017	2	2
3. Specific Requirements (Functional requirements)	07/10/2017	10/10/2017	3.5	1
5.1 Uml modelling (Use case)	08/10/2017	15/10/2017	5	8
5.2 Sequence diagram	16/10/2017	24/10/2017	12	9
3.4 Software system attributes	14/10/2017	15/10/2017	1	2
Final revision	25/10/2017	28/10/2017	5	9
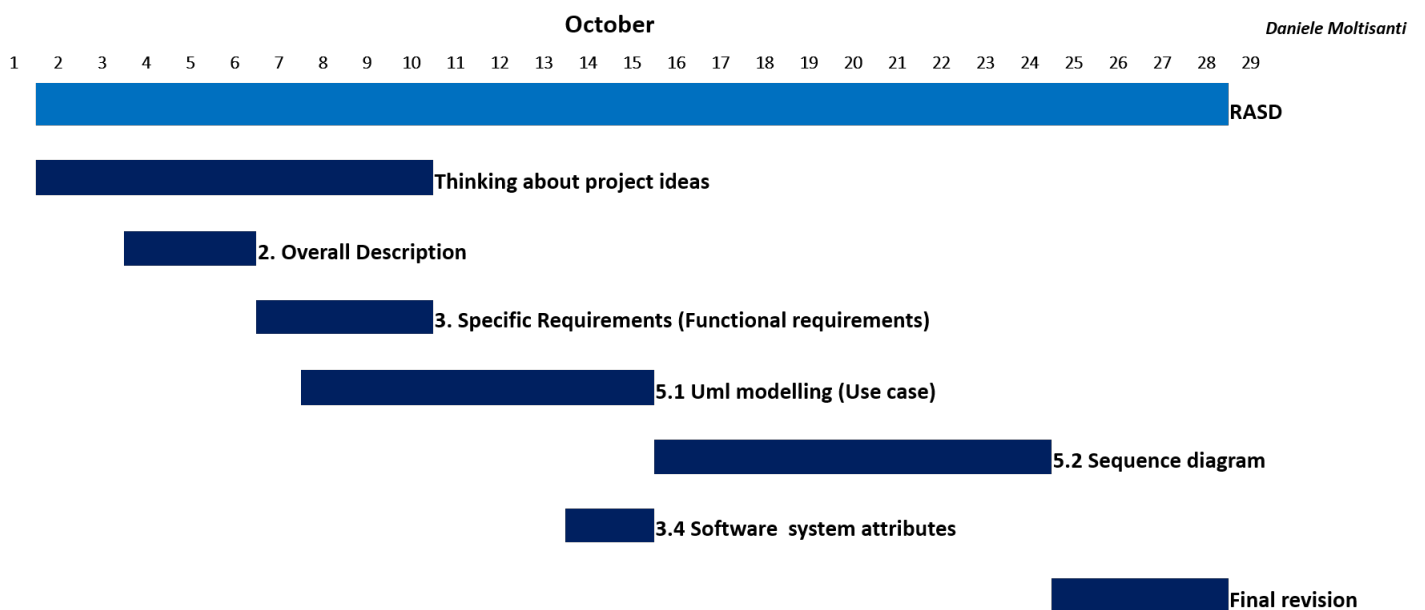
Figure 25: Gantt diagram Moltisanti