

POLITECNICO DI MILANO



**POLITECNICO**  
**MILANO 1863**

SOFTWARE ENGINEERING 2 COURSE

**Travlendar +**

*di*

*Gianluigi Oliva, Marco Mussi e Lukasz Moskwa*

October 27, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose . . . . .	4
1.2	Scope . . . . .	5
1.3	Definitions, Acronyms, Abbreviations . . . . .	5
1.4	Revision History . . . . .	7
1.5	Reference Documents . . . . .	7
1.6	Document Structure . . . . .	7
<b>2</b>	<b>Overall Description</b>	<b>8</b>
2.1	Product Perspective . . . . .	8
2.2	Product Functions . . . . .	9
2.3	User Characteristics . . . . .	9
2.4	Assumptions, Dependencies and Constraints	
	10	
<b>3</b>	<b>Specific Requirements</b>	<b>12</b>
3.1	External Interfaces Requirements . . . . .	12
3.2	Functional Requirements . . . . .	20
3.3	Performance Requirements . . . . .	50
3.4	Design Constraints . . . . .	50
3.5	Software System Attributes . . . . .	50
<b>4</b>	<b>Formal Analysis using Alloy</b>	<b>52</b>
<b>5</b>	<b>Effort Spent</b>	<b>63</b>
<b>6</b>	<b>References</b>	<b>64</b>

## **Abstract**

The main task of this document is to give a specification of the requirements that our system has to fulfil adopting the IEEE-STD-830-1993 standard for RASD documentation . It also introduces the functional and non-functional requirements via UML diagrams and a high level specification of the system. In the last part of this document it presents the formal model of the specification using Alloy analysis.

The information in this document are intended for the stakeholders and the developers of the project. For the stakeholders this document presents a description useful to understand the project development, meanwhile for the developers it's an useful way to show the matching between the stakeholders' requests and the developed solution.

# 1 Introduction

In this section we will explain which are the main scopes of the Travlendar+ application and we will provide a general overview of all the features.

## 1.1 Purpose

The main purpose of the RASD document (Requirement Analysis and Specification Document) is to highlight the domain in which the system will work and the primary use cases.

The document below specifies who will use the system, the reason for which the system is developed, what services will be provided and the environment in which the system finds its use. We will need to define the functional and not functional requirements of the system.

The main goals to be achieved are:

- The user wants to schedule his meetings
- The user wants to reach the place of his meeting with the most optimized path
- The user wants to know the weather forecast to be better organized for the journey
- The user wants to use the means of transport he prefers
- The user wants to purchase the tickets for the used means of transport
- The user wants to localize means of shared transport whenever they are around him
- The user wants to be sure there are no overlapping in his schedule
- The user wants to keep a range of time for his breaks
- The user wants to use only a specific kind of means of transport
- The user wants to modify his meetings and rearrange his schedule
- The user wants to share the time of his meetings

## 1.2 Scope

The scope of this application (Travlendar +) is provide a tool for the target users to schedule in an effective way their time optimizing the travels.

Travlendar+ allows to create a calendar which fits the meetings and other kind of commitments. The key feature of this application is to determine if the meeting location is reachable in the scheduled time and then provide a fast way to get to the destination, otherwise it notify the user that it is not possible by any mean to fulfill the request. Furthermore the application also arranges the schedule in a flexible way for some kind of events (like the lunch) and offers the possibility to slightly change its time.

In the creation of the user's calendar it also takes into account the current weather condition to make smarter decisions. As example, if the user show interest in using bike or a bike sharing service, during a sunny day, the system will favor a suitable path.

The user is invited to provide his order of preferences in terms of means of transport. In this way, he is free to choose if use bus, train or subway in a particular day. It is possible to select as well if you want to use a car or bike (your or a sharing service's one). The system also retrieves information about the current path situation like traffic jam or strikes and immediatly find alternative solutions.

With Travlendar+ is possible as well to buy public transport's tickets on the fly for the journey. If the user requires often the same path, he is suggested by the application to buy the best offered subscription.

## 1.3 Definitions, Acronyms, Abbreviations

### Definitions

- **Platform:** system/application as a whole.
- **User:** An end user who is currently registered to the Travlendar+ application and has credentials to access.
- **Guest:** Person not registered yet and with limited access to features.
- **Event:** A scheduled meeting or other kind of appointment a user has to attend.
- **Journey:** The path chosen by the application as the one with all the fulfilled requirements.
- **Bad Weather:** A weather that prevents the user from choosing some path options. The listed bad weathers are snow, rain, storm and others.
- **Framework:** Reusable set of libraries or classes for a software system.

- **Cross-Platform:** software able to run on different platforms with same code.
- **Port:** in the internet protocol suite, it is an endpoint of communication in operating system

## Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **DB:** Database
- **DBMS:** Database Management System
- **OS:** Operating System
- **HTML:** HyperText Markup Language
- **CSS:** Cascading Style Sheets
- **JS:** JavaScript
- **JSON:** JavaScript Object Notation
- **API:** Application Programming Interface
- **IDE:** Integrated Development Environment
- **RAM:** Random Access Memory
- **HTTP:** HyperText Transfer Protocol
- **HTTPS:** HyperText Transfer Protocol Secure
- **TCP:** Transmission Control Protocol

## Abbreviations

- **G<sub>n</sub>:** n-th goal
- **R<sub>n</sub>:** n-th functional requirement
- **D<sub>n</sub>:** n-th domain
- **M<sub>n</sub>:** n-th mockup
- **WebApp:** WebApplication

## 1.4 Revision History

Version, date and summary

Version	Date	Summary
1.0.0	October 27, 2017	First release of this document

## 1.5 Reference Documents

We used the following documents:

1. The original Travlendar application:  
<http://score-contest.org/2018/projects/travlendar.php>
2. The revised document of the assignment:  
<https://goo.gl/9m1ojy>
3. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
4. IEEE Std 1016tm-2009 Standard for Information Tecnology-System Design-Software Design Descriptions.

## 1.6 Document Structure

This document is composed by the following sections:

- **Section 1:** General overview of the Travlendar+ application and definition of the goals.
- **Section 2:** Overall description of the software feature and implemeted functions. Considerations about constraints, assumptions and dependencies.
- **Section 3:** Specification of functional and not functional requirements in a software and hardware perspective. Examination of design and performance.
- **Section 4:** Formal analysis of requirement using modeling language as Alloy.
- **Section 5:** Time and resource effort during the development of the application.
- **Section 6:** References and software used during the process of creation of the system.

## 2 Overall Description

### 2.1 Product Perspective

We are going to release a cross-platform web application which will be able to run on every device. This application won't provide an interface for the service administrator, because it is possible from the server side.

As we are developing a WebApp, we are willing to release custom API for any future application in order to facilitate further implementations.

#### World and Machine model interpretation

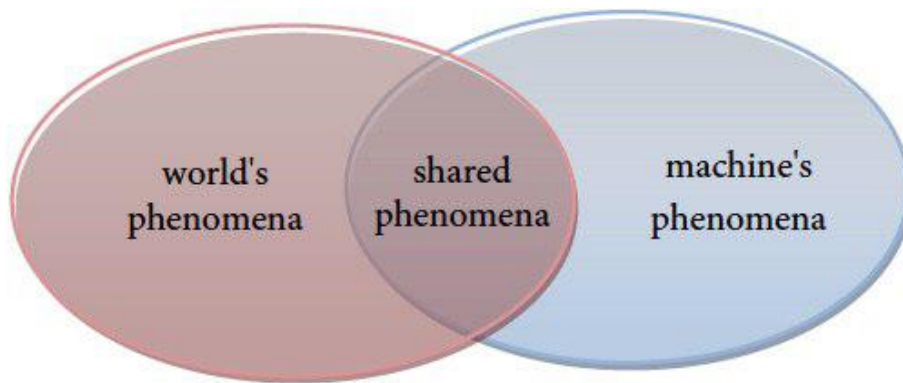


Figure 2.1: Relation between world and machine phenomena

From now on we will refer to **world** as everything that not concern our system and **machine** as everything about our system. Therefore **world's phenomena** are all the external events happening in the world and **machine's phenomena** are the events related to the system. There are as well some **shared phenomena** which are observable by both the parts.

In our case the world's phenomena are:

- Bus out of service or other mechanical failure in public transports
- Closed roads due to a public demonstration



In our case the machine's phenomena are:

- Estimation and managing of users' free time
- Database query
- User's registration

In our case the shared phenomena are:

- Tracking of public transport's means
- Strikes and public services manifestation

## 2.2 Product Functions

The product provide to users a simple and user-friendly interface to schedule their events and help them to organize their journey.

In particular the system is expected to be able to:

- Let the guest register and log in as users
- Schedule several events for every user
- Computate the best possible path for the daily journey
- Notify the user if the current journey is not realizable and why
- Interact with third part and sharing services
- Let the user buy tickets for the ride
- Support the user during the whole travel
- Allow the user to insert his preferred choice for the used transports
- Gives a real time evaluation of the environmental conditions (like strikes and weather situation) and uses it to find the best choice for the journey.

## 2.3 User Characteristics

We don't require the user to be particularly skilled in new technologies as we expect the users to be aware and capable of using basic functions of mobile devices.

The actors we identified are:

- **Guests:** people that haven't yet gained credentials to access the full service are only able to check the best journey options with some restrictions: they cannot save their schedule nor buy tickets or use third part sharing services.
- **Users:** guests that are currently registered to Travlendar+ and already verified their credentials (as e-mail address). Such users are able to log in and access full service with no restrictions.

## 2.4 Assumptions, Dependencies and Constraints

### Costraints

- **Hardware limitations:** Our application runs on every mobile device like smart-phones and tablets, independently of the OS (will run on Android, iOS and Windows Phone). Therefore, as the App consumes a low amount of RAM, the only hardware constraint for the users is to have a mid-range device. (for instance Nexus 4 or better for Android, iPhone 5 or better for iOS and Nokia Lumia 520 for Windows Mobile).
- **Interface to other applications:** The system needs to interfaces itself with other third part applications in order to provide the capability of buying tickets and booking sharing services.
- **Parallel operations:** The application must be able to handle multiple parallel requests from several users granting a high reactivity.
- **Availability limitations:** The system's uptime and availability are around the 98% corresponding to a maximum downtime of 3.36h per week.
- **Privacy Constraints:** The personal informations of the users, as for instance the current geo-position, must be safely stored and not reachable to other people.

### Assumptions and Dependecies

- **Internet Connection:** the device used by the users dispose of an internet connection and a sufficient bandwidth to use the application.
- **Device GPS:** the device used by the users dispose of a built in GPS Localizer.
- **Public transports tracking:** all the public transports are real time localized
- **No privileged users:** there are no priviled users or administrators with particular functions
- **No user connections:** every user is independent from the others, and their schedule does not affect by any mean the others schedules

## 2 Overall Description

- **Weather information not available:** when the weather informations are not available, the system computates the schedules without considering them
- **Booking range:** we assume that the users are not interested in planning events that will occur later than a week
- **City Location:** the users live and uses the application in the surroundings of Milan
- **Weather information provided:** we assume that the weather informations are correct and provided by a third part
- **Shared services provided:** we assume that the sharing services are working and provided by a third part
- **Host availability:** the server which hosts the application has an uptime greater or equal to the uptime of the application
- **Battery duration:** user's device's battery duration is enough to permit the full-fillment of the journey.
- **API availability:** the API provided by third part's services are always available
- **Break/Lunch time:** we assume that the user will eat near the same place of the event before the break.
- **OS Permission Granted:** the user will always grant to his OS's device the permission to access to all the needed services

## **3 Specific Requirements**

### **3.1 External Interfaces Requirements**

#### **User Interface**

The user interface must be user-friendly volted to guarantee to the final user an easy way to interact with the application. The development of the front end is realized due to HTML and CSS to provide a responsive interface for all devices.

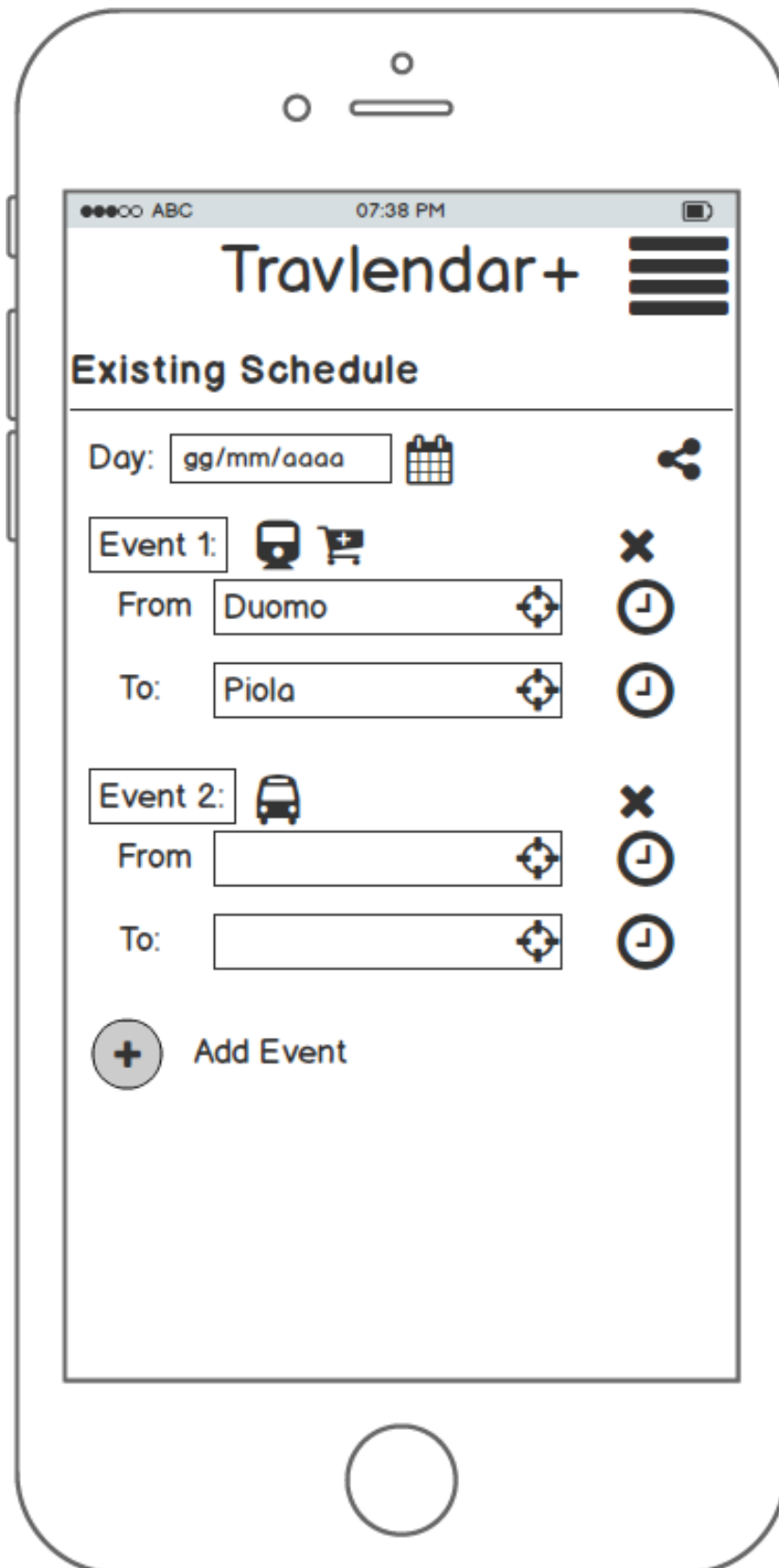


Figure 3.1: Mockup: In this interface the user can see his already created schedules and events

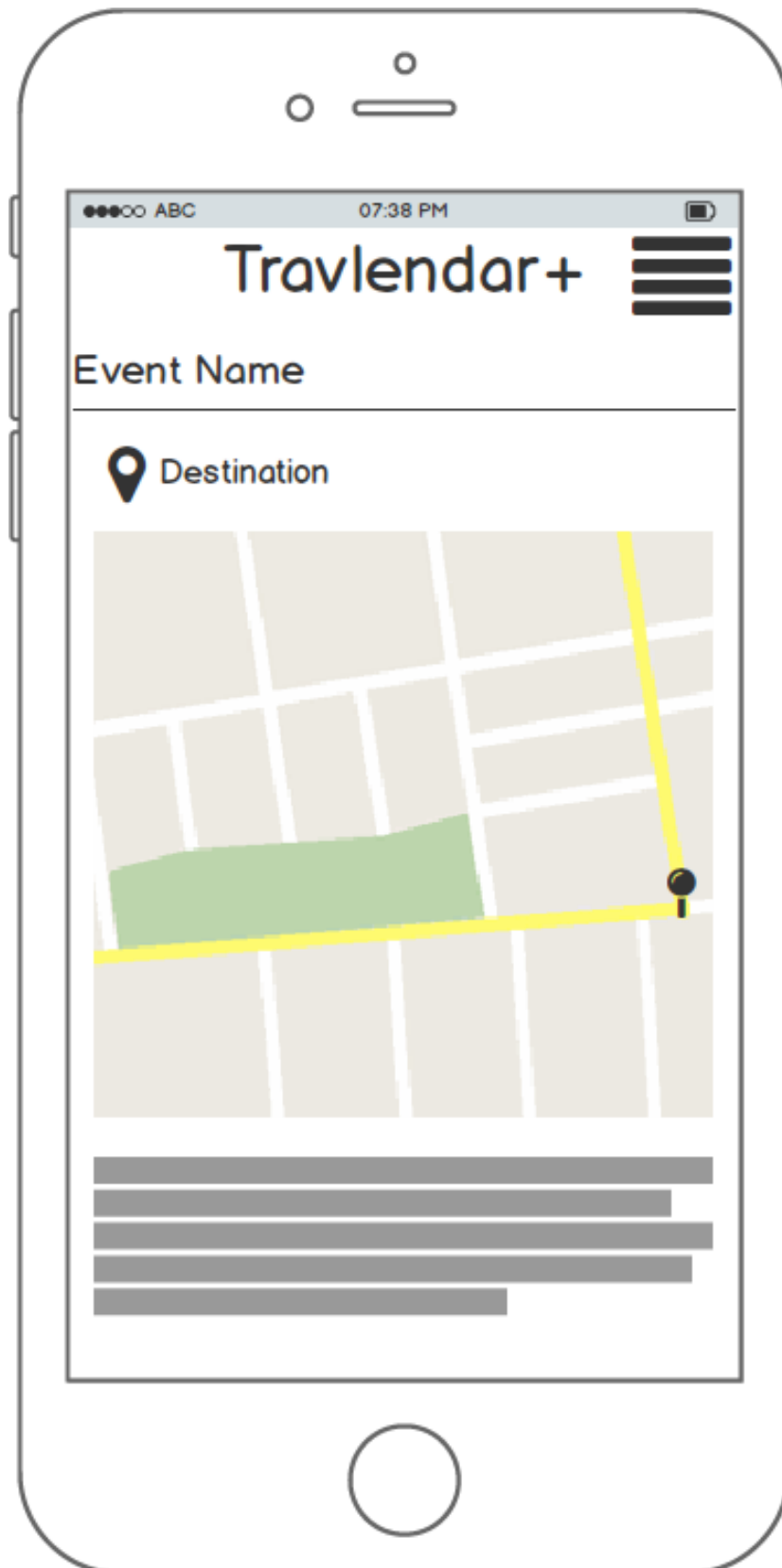


Figure 3.2: Mockup: This interface shows up to the user of the preview of his journey to reach the selected event

The image shows a mockup of a mobile application interface for 'Travlendar+'. The app is displayed on a smartphone frame. At the top of the screen, the status bar shows 'ABC' and '07:38 PM'. Below the status bar, the app's title 'Travlendar+' is displayed in a large, bold font, followed by a hamburger menu icon. The main heading of the screen is 'Register form'. The form consists of several input fields, each with a label and a placeholder text: 'Name:' with 'Enter name', 'Surname:' with 'Enter surname', 'e-mail' with 'Enter e-mail', 'Telephone:' with 'Enter telephone', 'Driving license' with 'Enter telephone', and 'Credit Card' with 'Enter credit card'. Below these fields is a checkbox labeled 'I agree to the Terms and Conditions of use'. At the bottom of the form is a 'Submit' button. The entire form is enclosed in a white box with a thin black border.

Figure 3.3: Mockup: This interface allow the user to perform the login action

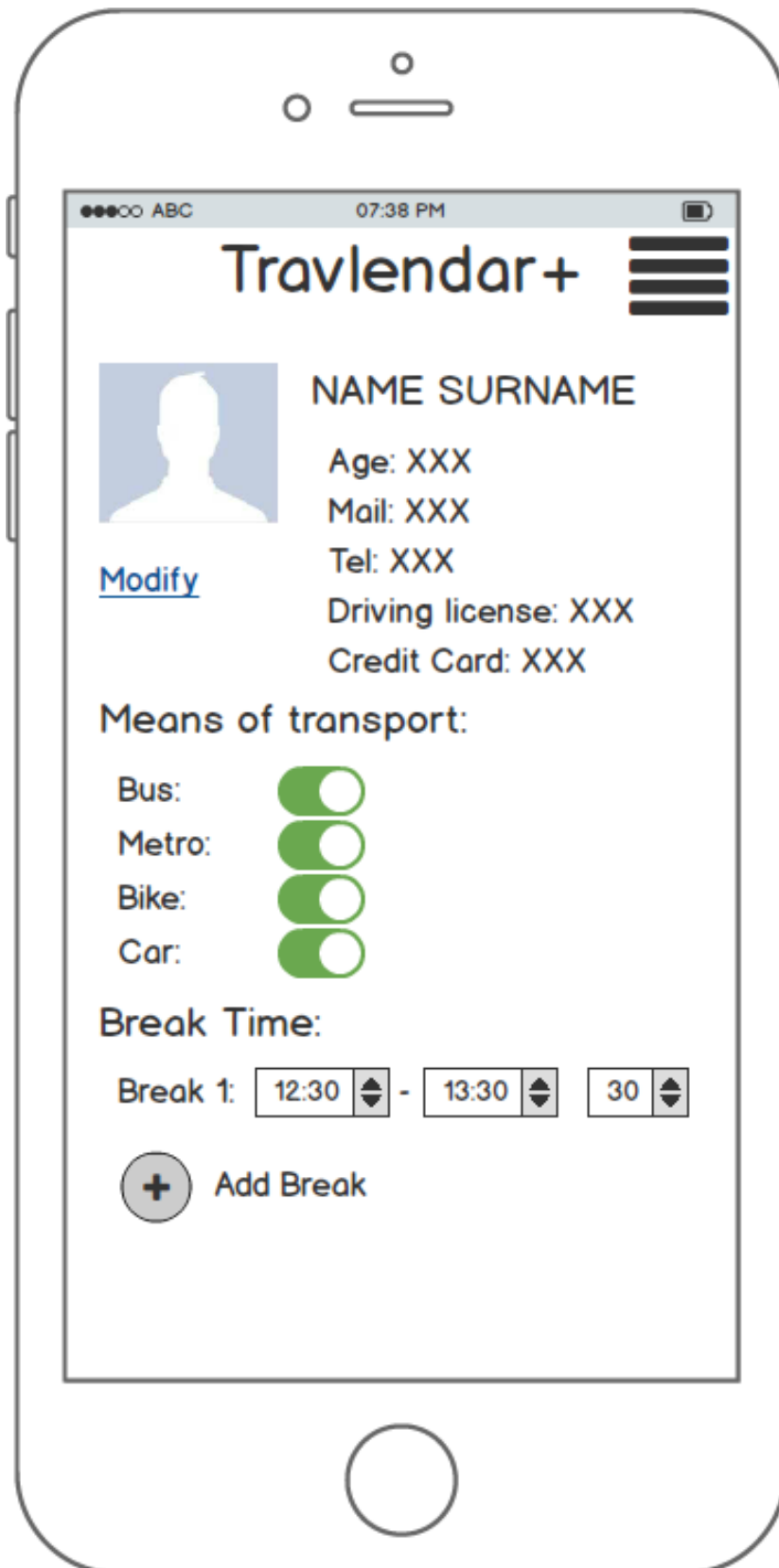


Figure 3.4: Mockup: In this section the user is able to view and modify his personal data



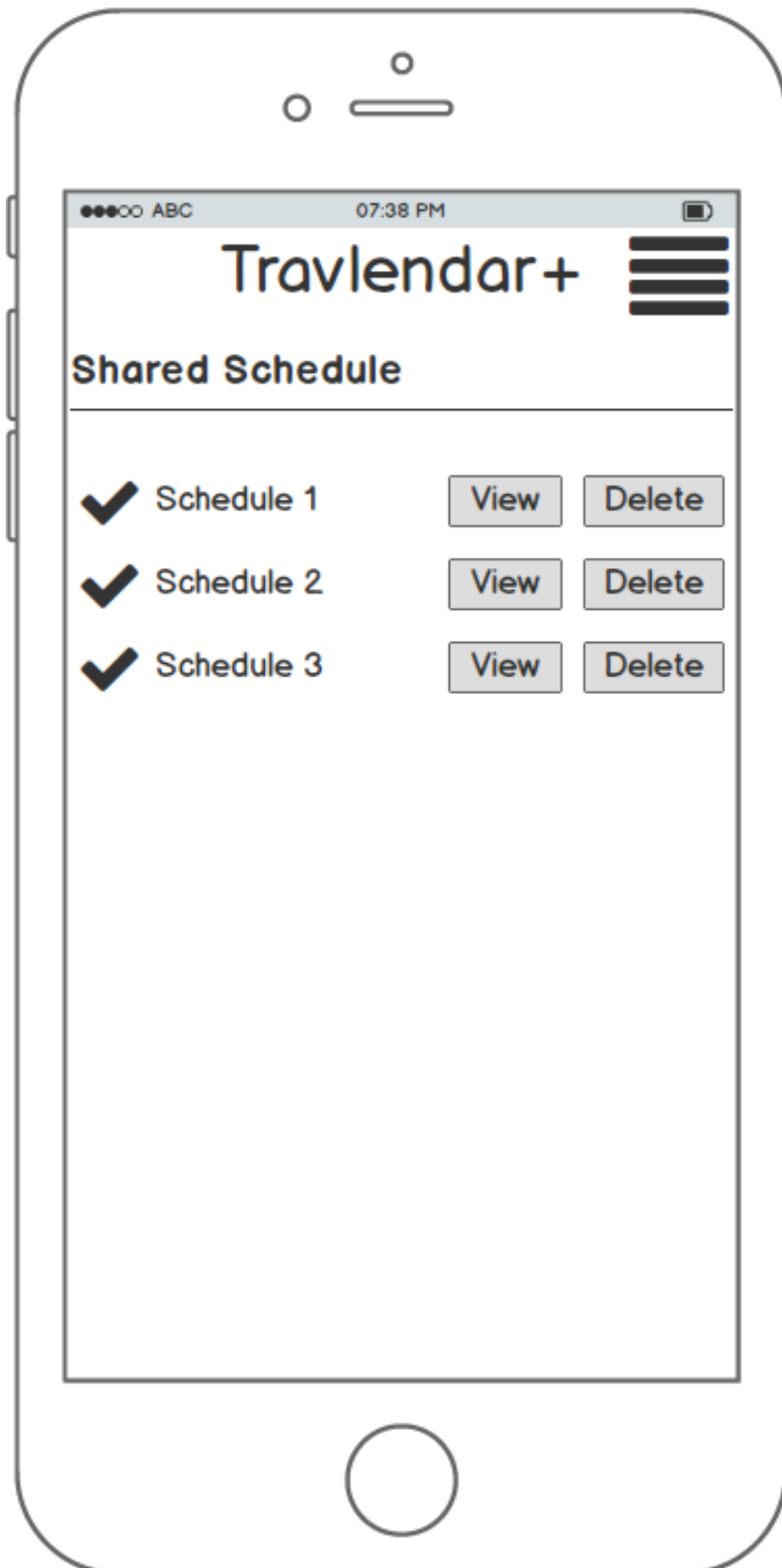


Figure 3.5: Mockup: This interface shows to the user the schedules that are actually shared with him

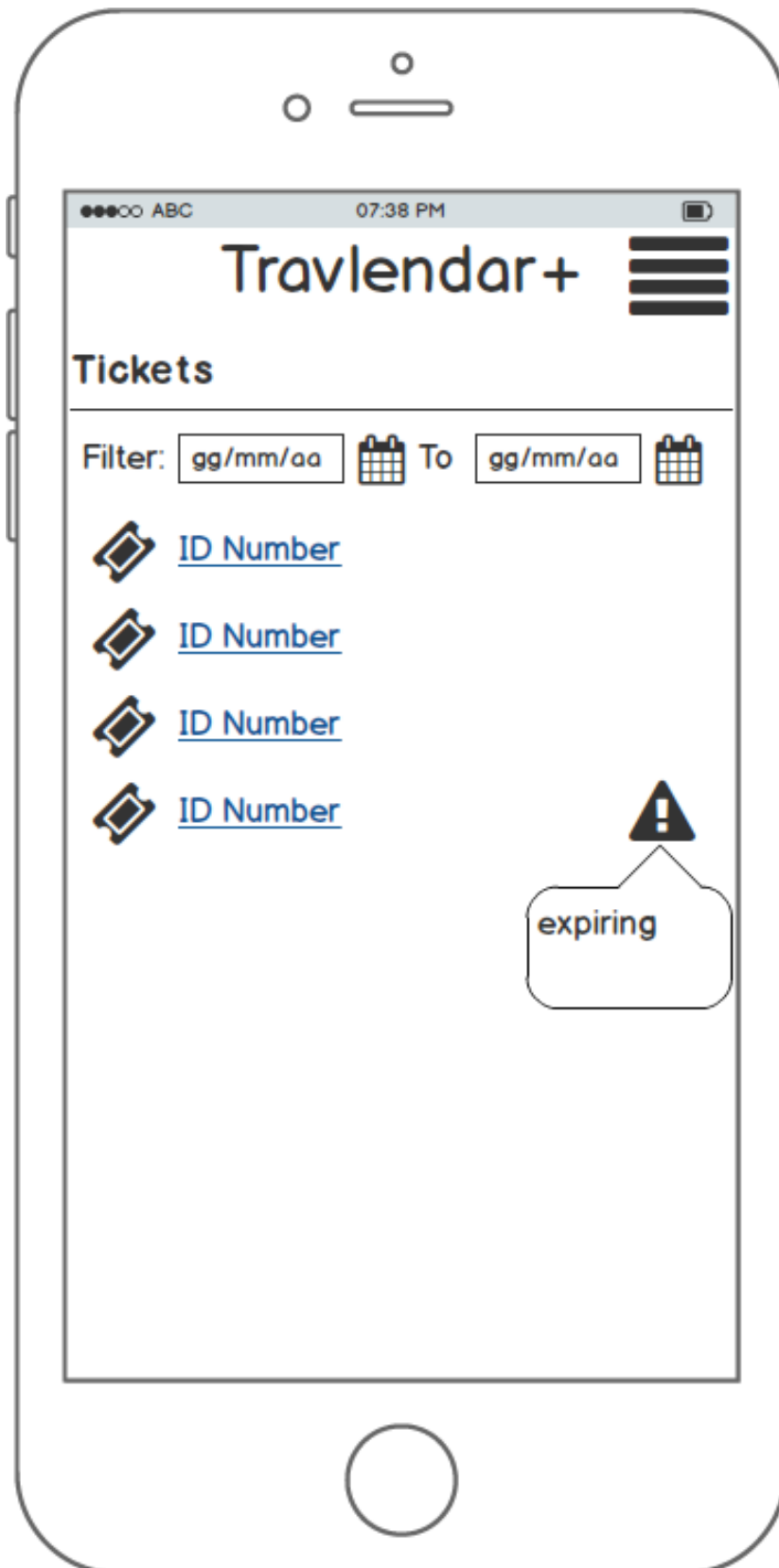


Figure 3.6: Mockup: In this section the user can see the tickets and subscriptions history

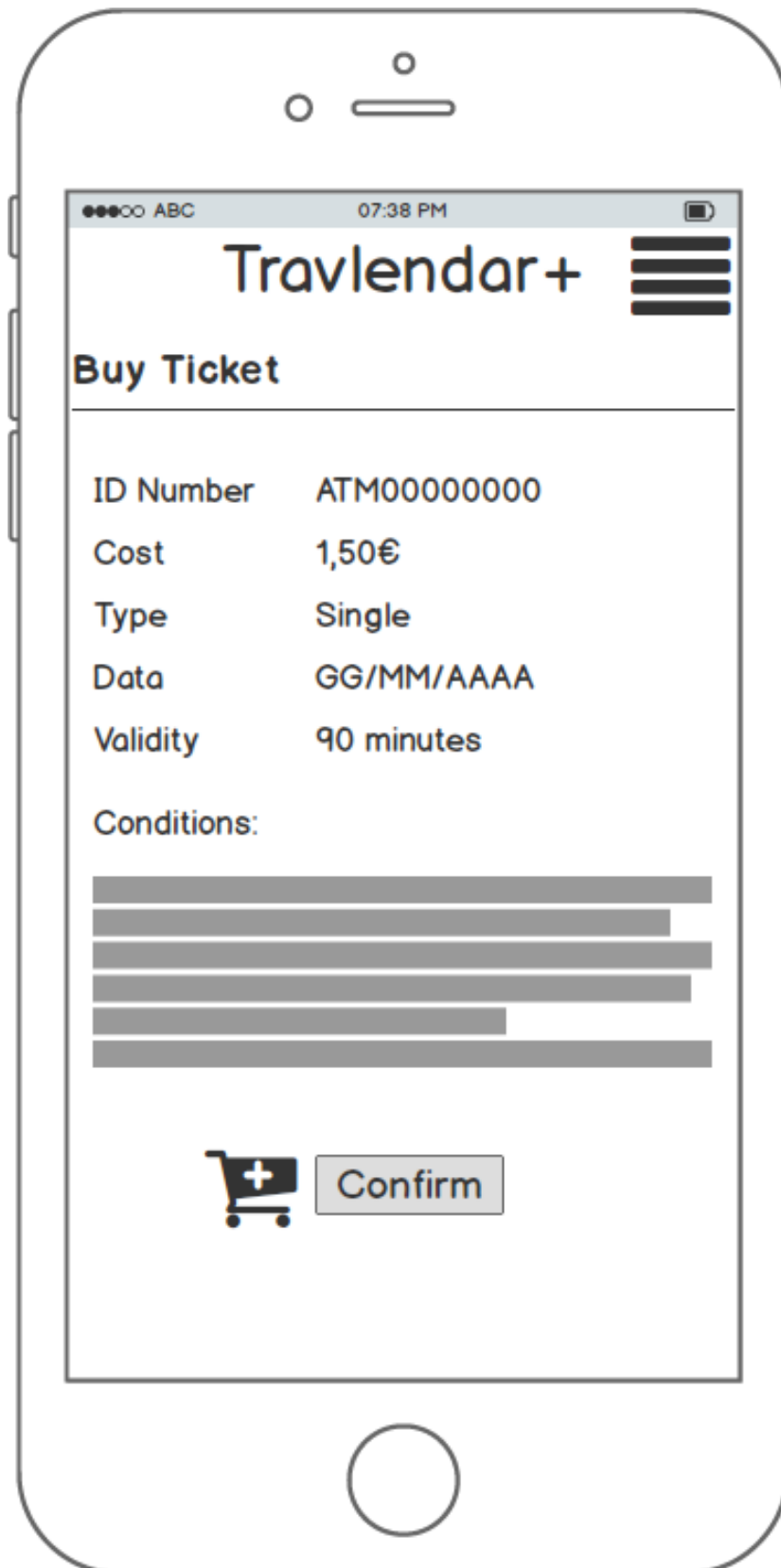


Figure 3.7: Mockup: This interface allow the user to perform the buy operation and complete his transaction

#### Hardware Interface

The system must be able to acquire the correct position of the means of transport and sharing services' vehicles. In order to acquire such data, every mean of transport must have a GPS localizer and internet connection to communicate its position to their server.

#### Software Interface

The system in order to be developed, both on the front-end side and back-end side, needs the use of the following programming, markup and representation languages:

- JavaScript
- HTML
- CSS
- Python
- SQL
- JSON

There are no particular limitation about the OS of the user's devices, as the application is cross-platform and can be also used with modern browsers.

#### Communication Interface

The communication between client and server happens using HTTPS by TCP protocol using port 443. A crypted channel is also used in order to perform secure transactions with third part sellers.

On the other hand the connection which allow to download public informations from third part API provided by major services like Google uses HTTP with port 80.

### 3.2 Functional Requirements

The user must be able to:

1. Register to Travlendar+ application.

The system:

- Provides a form to insert all the personal information like name, surname, e-mail...
- Verifies the genuineness of the e-mail address by sending an e-mail with a confirm request

### 3 Specific Requirements

2. Create a daily schedule specifying the location in which the event will take place and its timetable.

The system:

- Provides a form to be fulfilled with all the information about the event
- Checks that all the relevant fields were compiled in order to approve the request
- Provides a method to prove that the request was confirmed
- Deletes all unsaved changes to the planning if the internet connection is absent or the application server is down
- Finds the best route solution starting from some user's parameters
- Notifies the user whenever the given parameters lead to an impossible solution
- Takes in account the weather condition when looking for the best mean of transport and provides the options that can be selected by the user
- Track the user initial position through the GPS or an inserted address
- Notifies the user if there is a lack of internet connection and therefore a restriction to the application's services

3. Manage his own data.

The system:

- Provides a method to change all the user's data like e-mail, telephone number..
- Requires another e-mail verification in case the original e-mail was changed
- Let the user modify his preferences about the desired means of transport
- Let the user set or modify his preferences about the desired break time interval and how much it lasts

4. Retrieve informations about the weather forecast in a specified range of dates

The system:

- Allows the user to select the date to check the weather in
- Retrieves the data about the weather forecast
- Uses the GPS position in order to retrieve the informations about the city the user is requesting the forecast for

5. Buy tickets and subscription for public transports.

The system:

- Provides the user all possible choices to perform a payment (like PayPal, Credit Card and other kinds of money transfer)
- Notifies the user, whenever a fail occurs, the reason why it failed
- Runs along with third parts applications to fulfill the request

### *3 Specific Requirements*

- Memorizes all the bought or already owned tickets or subscriptions
  - Provides an easy and accessible way to visualize already bought tickets
  - Notifies the user if any of the subscription are evaluated
6. Share his own daily schedule with other users.  
The system:
- Provides an easy way to share datas with other users
  - Saves the received shared schedule to be seen later
7. Modify an already compiled schedule for a certain day.  
The system:
- Provides a method to allow the user to reschedule the timetable with new preferences
  - Notifies the user whenever his changes lead to a non optimal or impossible solution
  - Let the user delete an already created schedule
8. Choose his preferred time for having a meal.  
The system:
- Let the user decide the range of time when he wants to eat and the preferred time
9. Receive information about the current journey.  
The system:
- Provides all the informations about the selected journey showing the way to travel
  - Shows any information about the current traffic jam and road works
  - Informs the user about the waiting time for public transports

## Scenario, Use case diagrams and Flow Diagram

### Scenario 1

Mark has already bought a new smartphone so he decide to change the application to manage his timetable with the scope to optimize his schedule. So he decides to sign up Travlendar+ Application from his smartphone. First of all he downloads the application from the smartphone's store and, once is installed on the device, he fills all the field the registration form in the correct way and, once he confirms the authenticity of the e-mail address, he can join all services provided by the application.

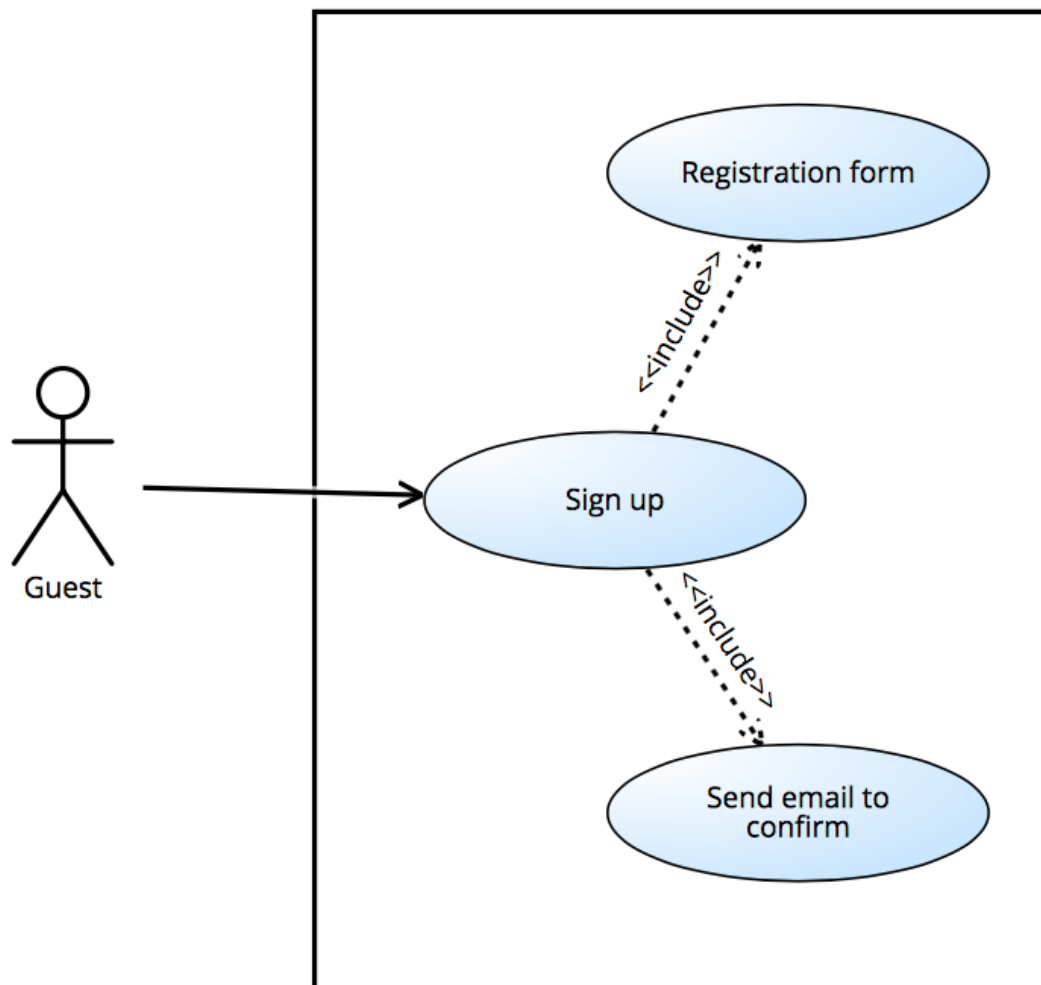


Figure 3.8: Use Case: Sign up

### 3 Specific Requirements

<b>Name</b>	Sign up
<b>Actor</b>	Guest
<b>Entry Condition</b>	The actor hasn't got an account on the system
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The guest download the application and open the application</li><li>• The guest clicks on sign up button</li><li>• The application provides the registration form</li><li>• The guest inserts mandatory data in the form like name, surname,e-mail.</li><li>• The guest optionally he could provides optional data like phone, driving licence number and the credit card to eventually pay the services linked to the app</li><li>• The guest agrees about terms and conditions of the service by put a tick in the checkbox</li><li>• The guest clicks submit button</li><li>• The system sends entered data to the server</li><li>• The system sends to his e-mail address a mail with a link to check the e-mail address</li><li>• The user clicks on the provided address</li><li>• The system validates the email address and the account</li></ul>
<b>Exit condition</b>	The actor become a user of the system and he can use the application.
<b>Exceptions</b>	Some mandatory field are not correctly filled and the system ask to the guest to correct it or the mail is not correct so is not possible to complete the sign up and the guest must redo the operation filling the field corresponding with the correct e-mail address. Also if he don't agree with terms and condition of the service, the application provide an error message and ask to check the corresponding box to complete the operation.



### 3 Specific Requirements

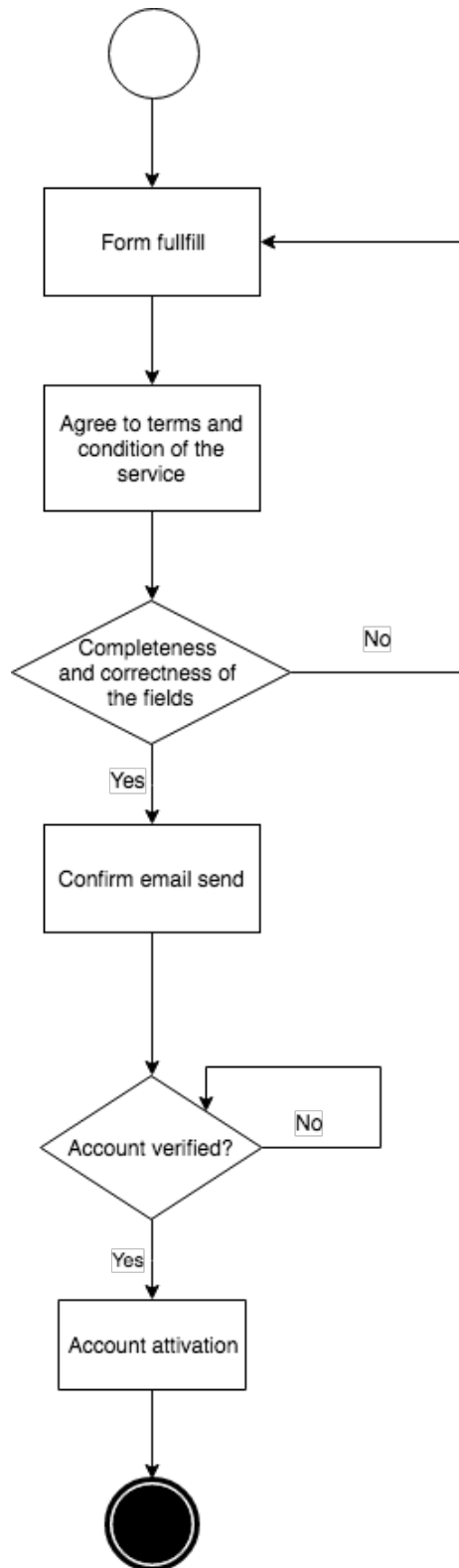


Figure 3.9: Activity Diagram: Sign up

## Scenario 2

Susan performs the login from her own mobile device because she wants to schedule a work's meeting that she will attend near Cairoli. Immediately after she has to move near Navigli to attend her friend birthday party. In order to do so she access to the event creation's dedicated section completing the proposed form with all the requested informations about the meeting (like the time it starts and ends). Then, the system calculates every possible journey solution based on the user's previously inserted preferences and the weather conditions. After she completes the creation's procedure selecting the best means of transport, Susan insert as well the informations about the birthday party and performs the same operations.

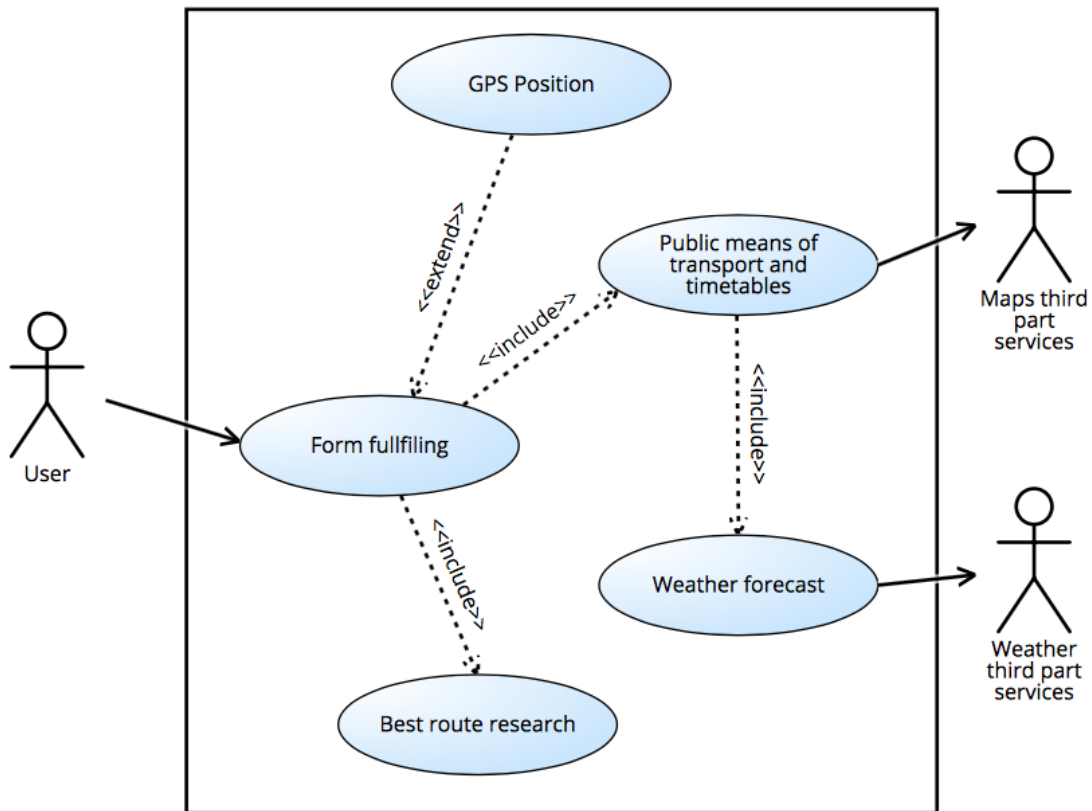


Figure 3.10: Use Case: Schedule creation

### 3 Specific Requirements

<b>Name</b>	Schedule creation
<b>Actor</b>	User, Weather third part services, Maps third part services
<b>Entry Condition</b>	The user is already logged in with his username and password
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user select the "Schedule creation" section from the lateral menu</li><li>• The user fullfill the form selecting the event date, time and the starting location (using an address or clicking the icon for using GPS position) and the destination location</li><li>• The user press the confirm button</li><li>• The system request the information about all the possible routes to calculate to Maps third part services</li><li>• The system calculates the best route based on the user's previously selected preferences</li><li>• The system shows to the user the proposed solution</li><li>• The user confirms the schedule</li><li>• The system stores in the database the selected schedule for the user</li></ul>
<b>Exit condition</b>	The user creates a schedule that supports him in reaching on time his destinations
<b>Exceptions</b>	Some exception can occur: if the user doesn't fill all the form's fields the system notifies it and ask the user to compile them again. If the destinations location is not reachable in the given time or there isn't any possible route traceable the system notifies the user and delete the current operation. The user refuses the created schedule and the system deletes the current operation. If the internet connection is dropped while the operation is ongoing the system deletes the operation and notifies the user.

### 3 Specific Requirements

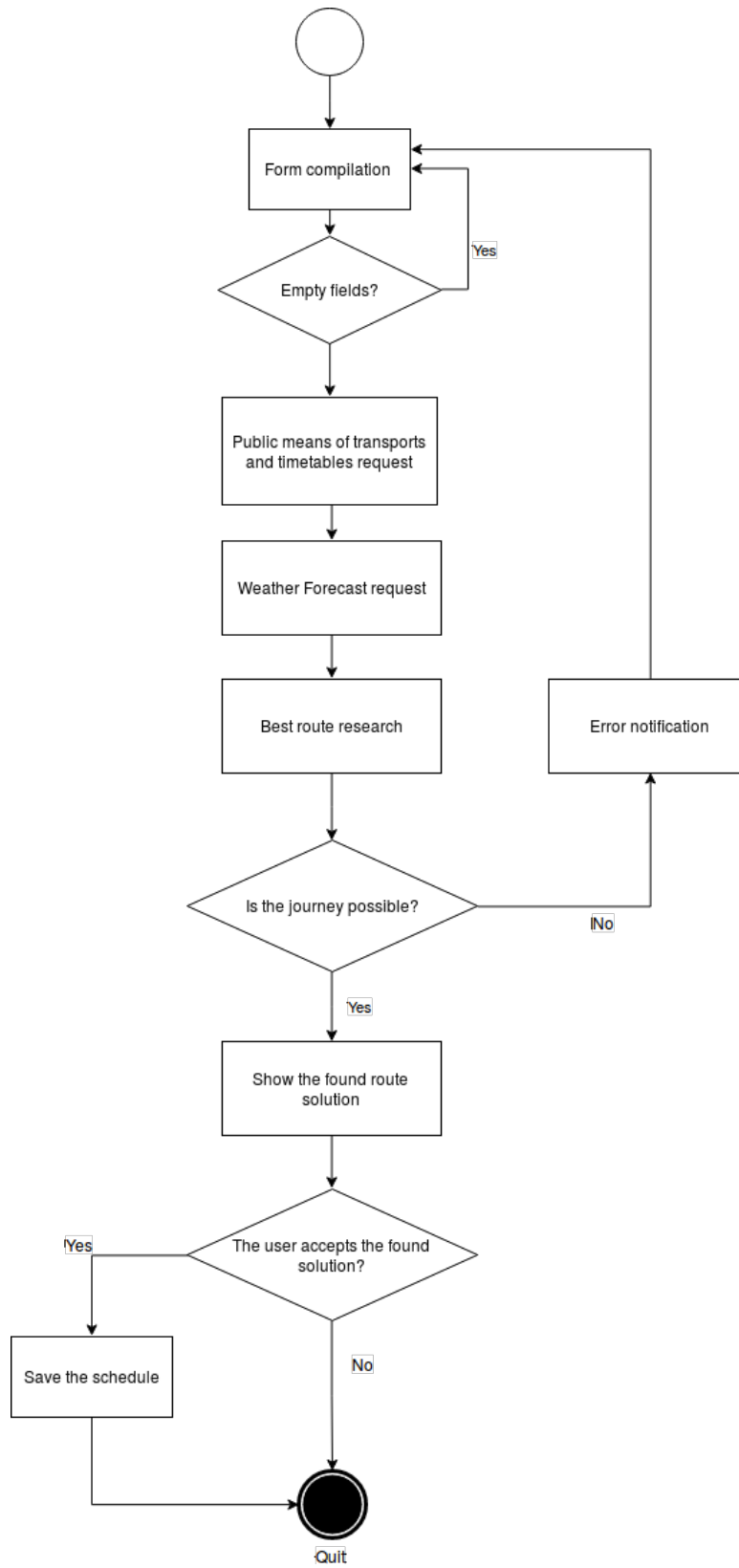


Figure 3.11: Activity Diagram: Schedule creation

### Scenario 3

Mark hear from her friend Susan that the application could be personalized with the purpose to better adapt to the user needs so he decides to look at the user's informations page and modify his profile to adapt it. He hate to use the bus because he thinks it is too slow and require a lot of time to reach the destination using so he switch off the corresponding field.

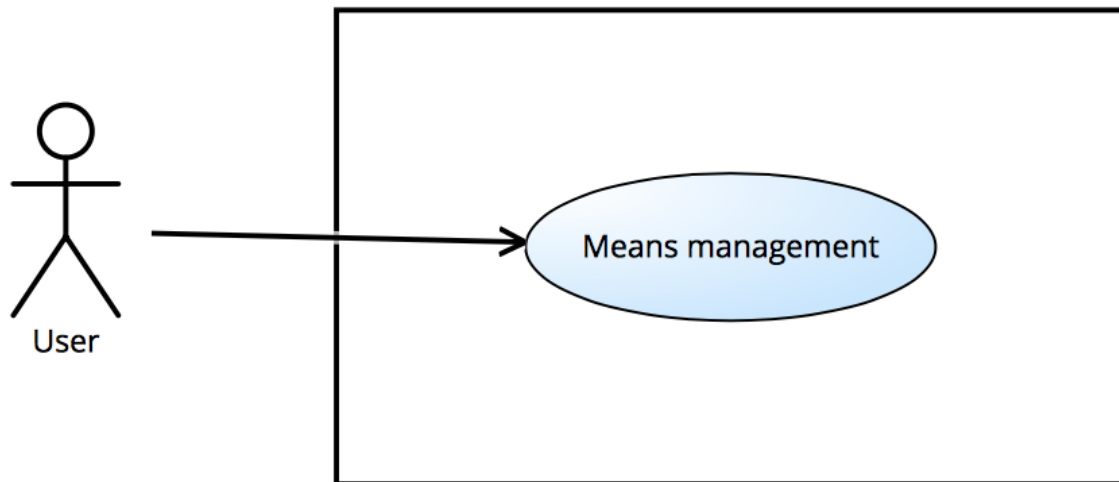


Figure 3.12: Use Case: Manage his preference about means of transport

### 3 Specific Requirements

<b>Name</b>	Manage his preference about means of transport
<b>Actor</b>	User
<b>Entry Condition</b>	The user already logged want to select his preferences about means of transport
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user clicks on the menu button and find the page about the account management</li><li>• The system provides all the information that it know about the user and permit to modify all data defined editable.</li><li>• The user find the switch corresponding to the bus option and turn off it</li><li>• The system update the user preferences</li></ul>
<b>Exit condition</b>	User setted the preferences about the bus
<b>Exceptions</b>	The user turn off all the switch so the application return an error message

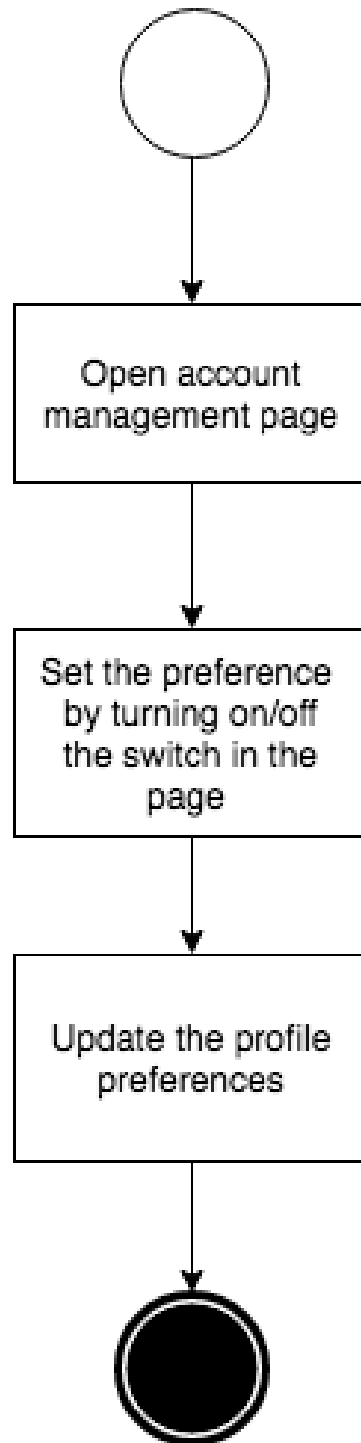


Figure 3.13: Activity Diagram: Manage his preference about means of transport

## Scenario 4

Susan performs the login operation from her own mobile device because she wants to check the Monday's weather conditions and know how to dress for her city festival. In order to do so she access the "Weather Forecast" section from the lateral menu and insert the date she is interested in. The system the requests Weather third part services' informations and shows the weather conditions.

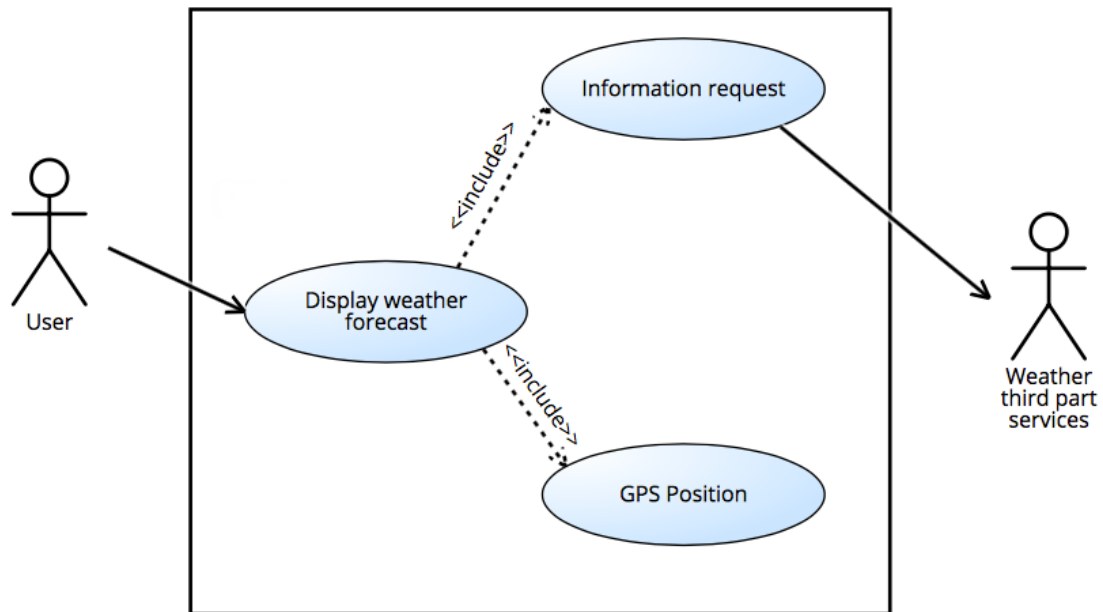


Figure 3.14: Use Case: Weather Forecast consulting



### 3 Specific Requirements

<b>Name</b>	Weather Forecast consulting
<b>Actor</b>	User, Weather third part services
<b>Entry Condition</b>	The user already logged want to select his preferences about means of transport
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user selects the "Weather Forecast" section from the lateral menu</li><li>• The user inserts the date of the day he is interested retrieving information in</li><li>• The system uses the GPS to determine the current user's current position</li><li>• The system interacts with Weather third part services to request the information about the weather forecast</li><li>• The system shows to the user the information received</li></ul>
<b>Exit condition</b>	The user obtains the informations about the forecast required
<b>Exceptions</b>	Some exceptions can occur if: the user inserts a date which is over a certain range. If the GPS position is not available the system notifies the user with an error message. If the weather forecast information are not available the system notifies the user.

### 3 Specific Requirements

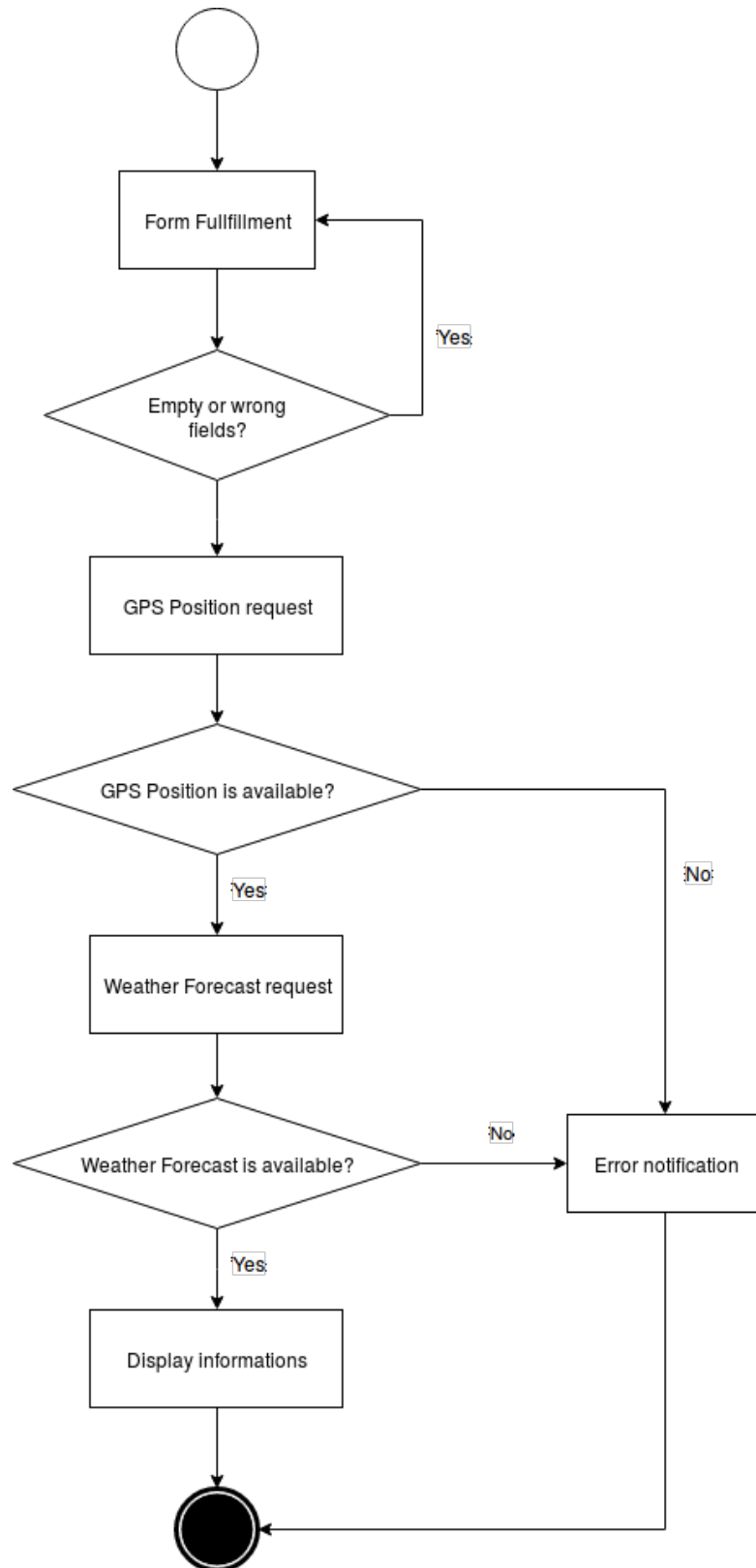


Figure 3.15: Activity Diagram: Weather Forecast consulting

## Scenario 5

John is a common Travlendar+ user and very often needs the application in order to schedule his appointments during the day. Sometimes he is so busy that he actually forgets to renew his weekly bus subscription. John isn't worried about that, because he knows that he can do it manually using Travlendar+ which will automatically allow him to choose his preferred payment method and perform the transaction when he creates the event. After the transaction, John wants to be sure that the ticket is actually there, so he checks his Tickets section, where all his tickets are stored.

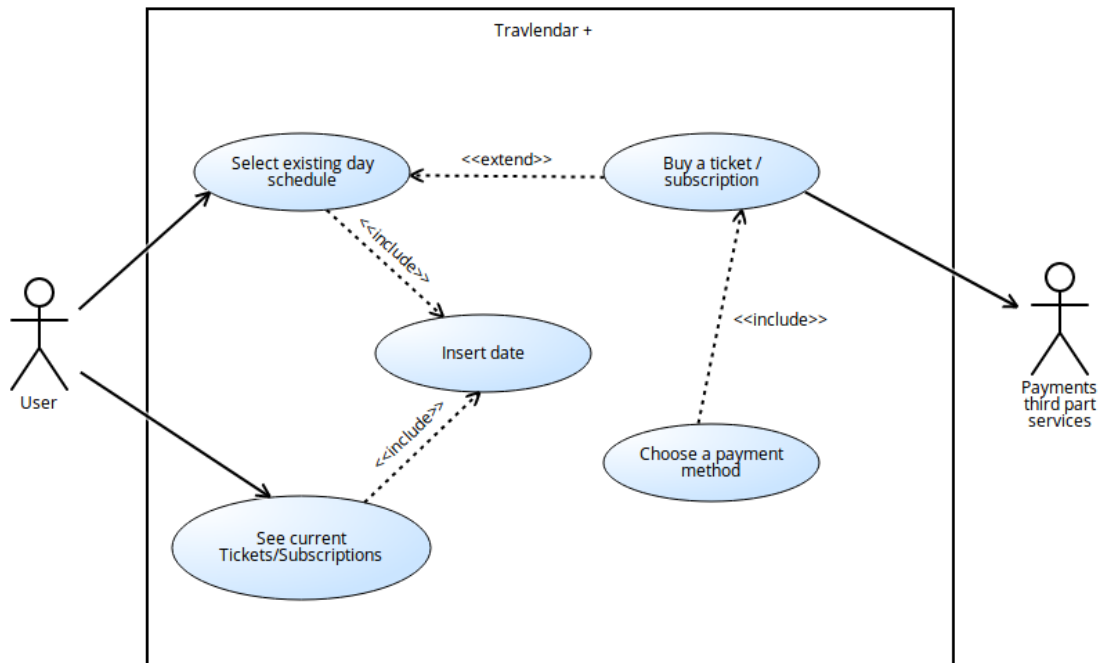


Figure 3.16: Use Case: Buy tickets and subscription for public transport

### 3 Specific Requirements

<b>Name</b>	Buy tickets and subscription for public transport
<b>Actor</b>	User, Google third part services, Payments third part services
<b>Entry Condition</b>	The user already logged has to renew on the fly his subscription or buy an occasional ticket
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user selects from the lateral menu the Existing Schedule section</li><li>• The user selects the date of the schedule he wants to buy tickets/subscriptions for</li><li>• The user tap the cart icon if he wants to perform a purchase</li><li>• The user choose a payment method</li><li>• The system requires a form for fullfilling the payment transaction</li><li>• The user complete the form requested</li><li>• The user press the confirm button</li><li>• The system interact with the payments third part service to</li><li>• complete the transaction</li><li>• The system store the receipt on the database</li><li>• The system notifies the user that everything went well</li><li>• The user selects from the lateral menu the Tickets section</li><li>• The system shows all the history of payments/tickets</li></ul>
<b>Exit condition</b>	The user perform the transaction and get the subscription/ticket
<b>Exceptions</b>	The user compiles in a wrong way the given form. The ticket selected is not purchasable anymore. There are errors in the payments third part services

### 3 Specific Requirements

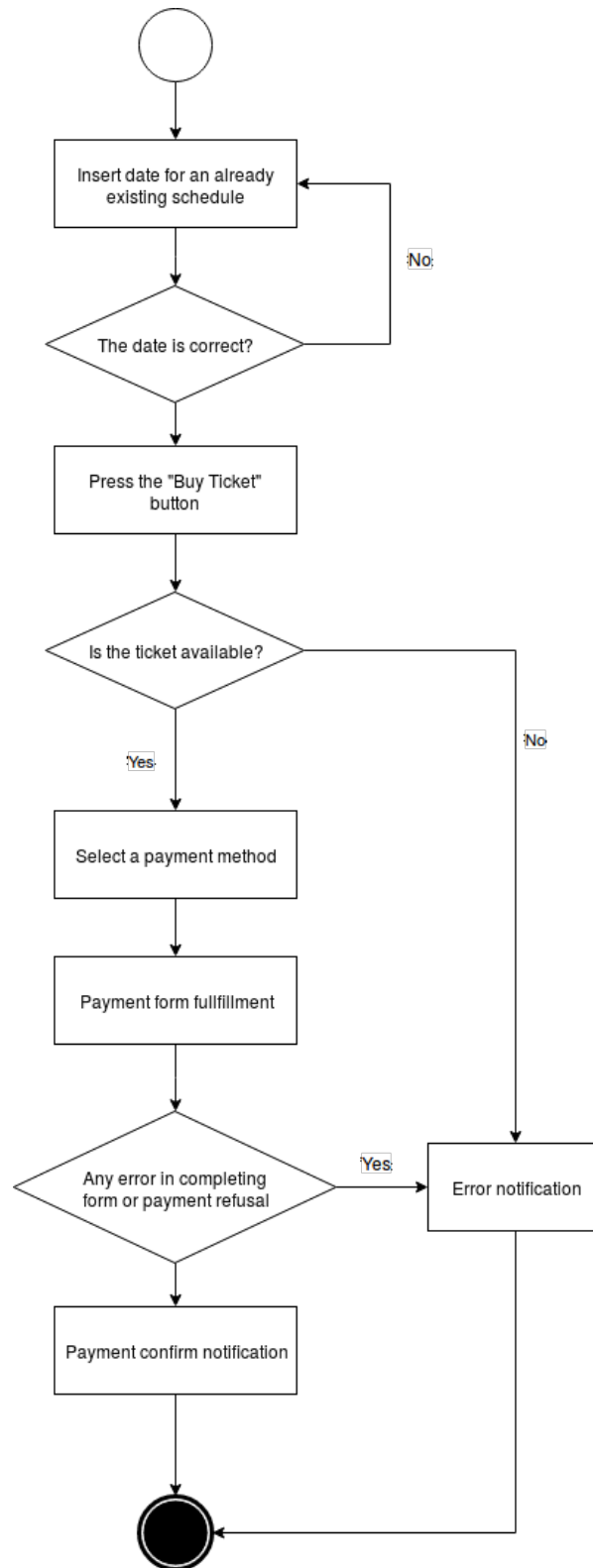


Figure 3.17: Activity Diagram: Buy tickets and subscription for public transport

## Scenario 6

John is a really busy man, and sometimes he has even like 5 events during the day. Whenever a colleague of his, which is a Travlendar+ user as well, asks him "What are you doing tomorrow?" he doesn't really remember. But John knows that with Travlendar+ it is actually easy to share his daily schedule with other users, so he decide just to send them all his events of a given day.

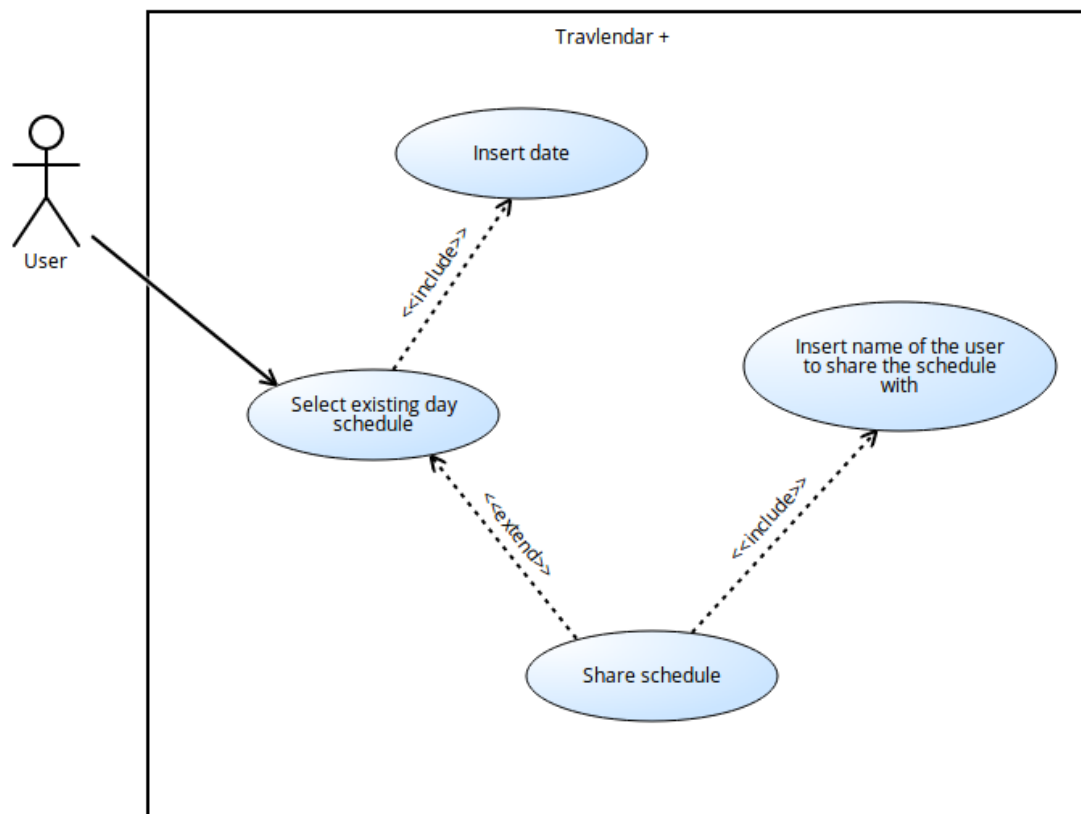


Figure 3.18: Use Case: Share user's his daily schedule with other users

### 3 Specific Requirements

<b>Name</b>	Share user's his daily schedule with other users
<b>Actor</b>	User
<b>Entry Condition</b>	A user already logged wants to share his daily schedule with another user
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user selects from the lateral menu the Existing Schedule section</li><li>• The user selects the date of the schedule he wants to share</li><li>• The user select the share icon</li><li>• The user writes the username of another existing user on Travlen-dar+</li><li>• The system searches the user associated with the inserted user-name</li><li>• The user confirms that he wants to share his daily schedule</li><li>• The system loads the schedule into the database's table associated to the other user</li><li>• The system notifies that the operation has concluded with success</li></ul>
<b>Exit condition</b>	The user shares his daily schedule of a given day with another user
<b>Exceptions</b>	The user insert the wrong name of another username, which notifies that the user doesn't exist. The user inserts a date where there are no schedule already saved. If the internet connection drops while the user is performing such operation or the system is not reachable, the operation gets deleted.

### 3 Specific Requirements

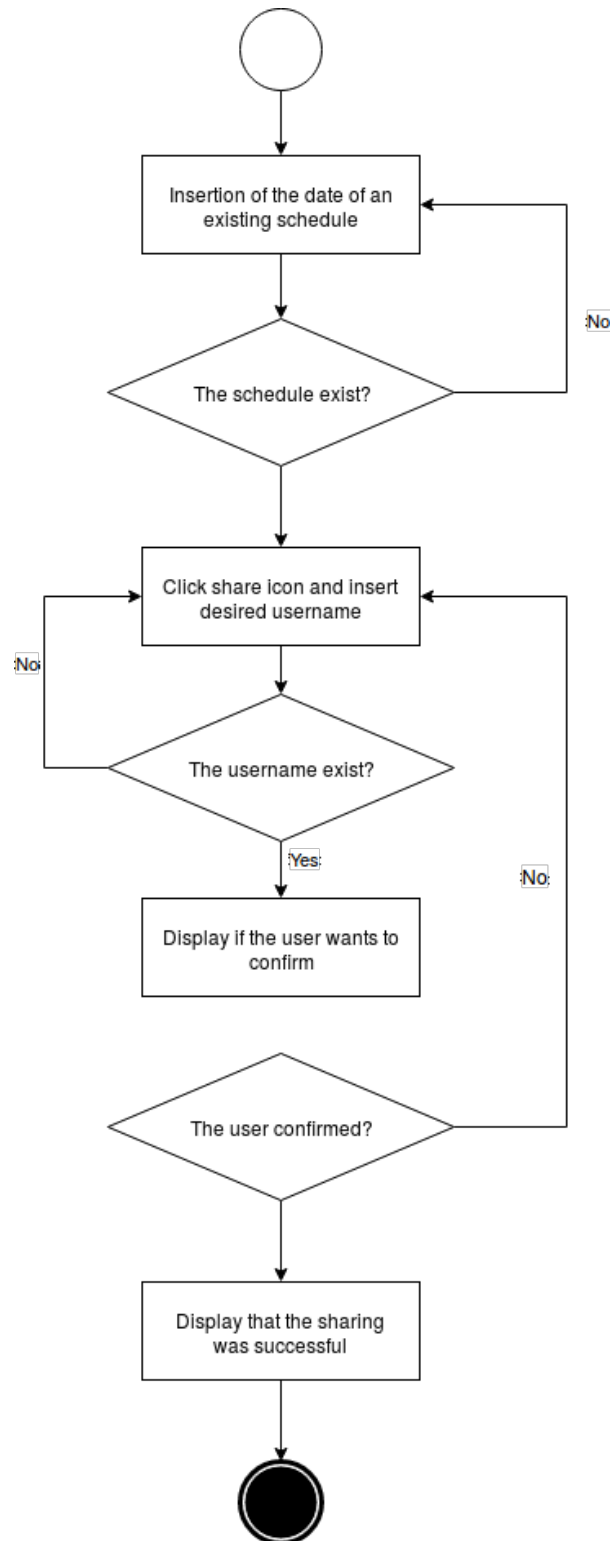


Figure 3.19: Activity Diagram: Share user's his daily schedule with other users



## Scenario 7

Mark wants to reschedule the appointment of the 3pm putting it forward of an hour and wants to know if it is possible to reach the destination on time and also if it grants his lunch. He already scheduled the appointment on Travlendar+ so he is looking for moving it an hour before. So he opens Travlendar+ calendar application, select the event and move. He sees by the application that it is possible and confirm to Anne the moving of the meeting.

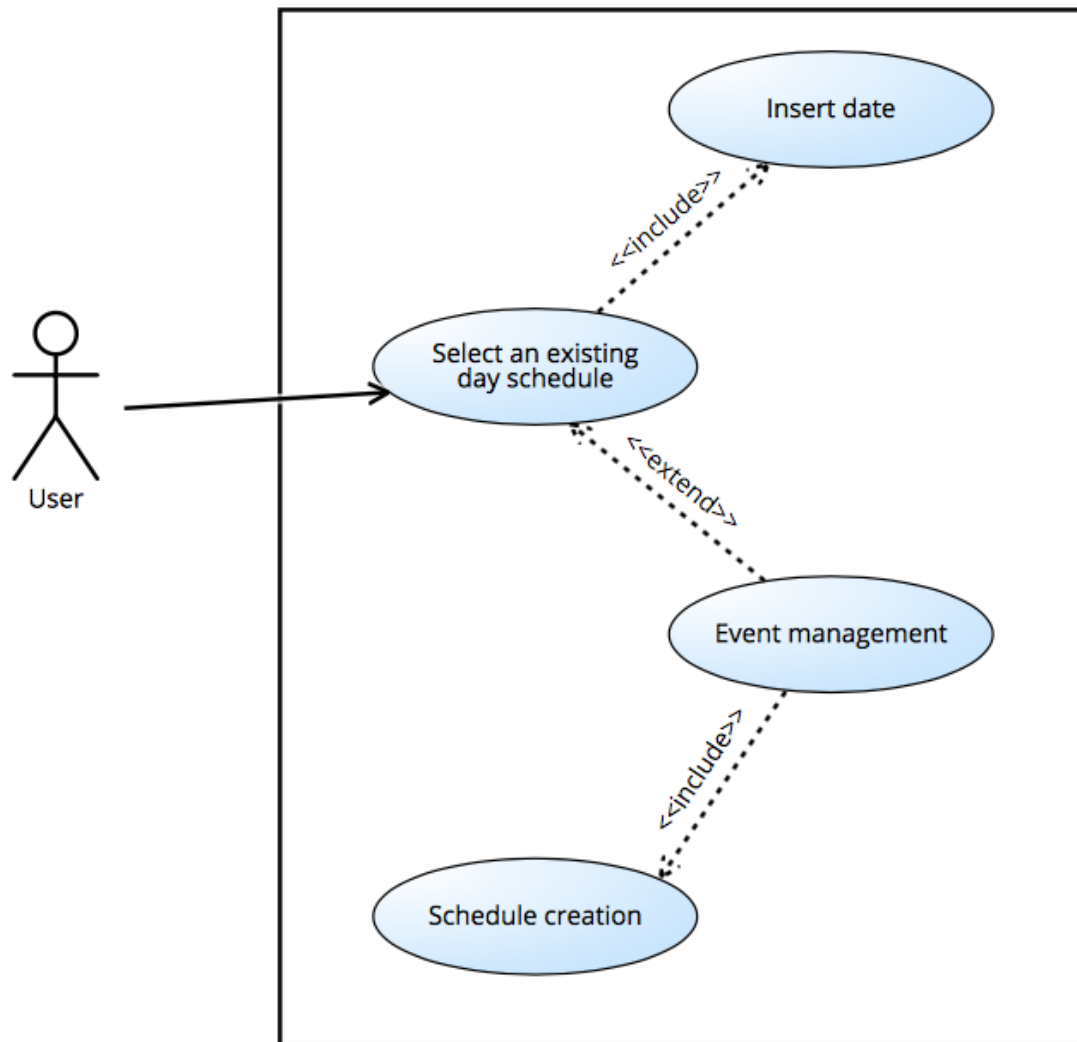


Figure 3.20: Use Case: Modify an already compiled schedule for a certain day.

### 3 Specific Requirements

<b>Name</b>	Modify an already compiled schedule for a certain day.
<b>Actor</b>	User
<b>Entry Condition</b>	The user is already logged in and he has already fill his timetable with al least one appointment
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user selects from the lateral menu the Existing Schedule section</li><li>• The user insert the data of the event he wants to modify</li><li>• The system shows all the events of the day</li><li>• The user clicks on the event he wants to move</li><li>• The user inserts the updated information</li><li>• The user confirms the choice</li><li>• The system calculates the new journey</li><li>• The system returns a positive feedback so the event updated</li></ul>
<b>Exit condition</b>	The user succeed in updating his schedule
<b>Exceptions</b>	The application report an error if the schedule is not feasible. If the user inserts wrong data the system notifies the error. If the selected data does not contains an event the user is notified of the error.

### 3 Specific Requirements

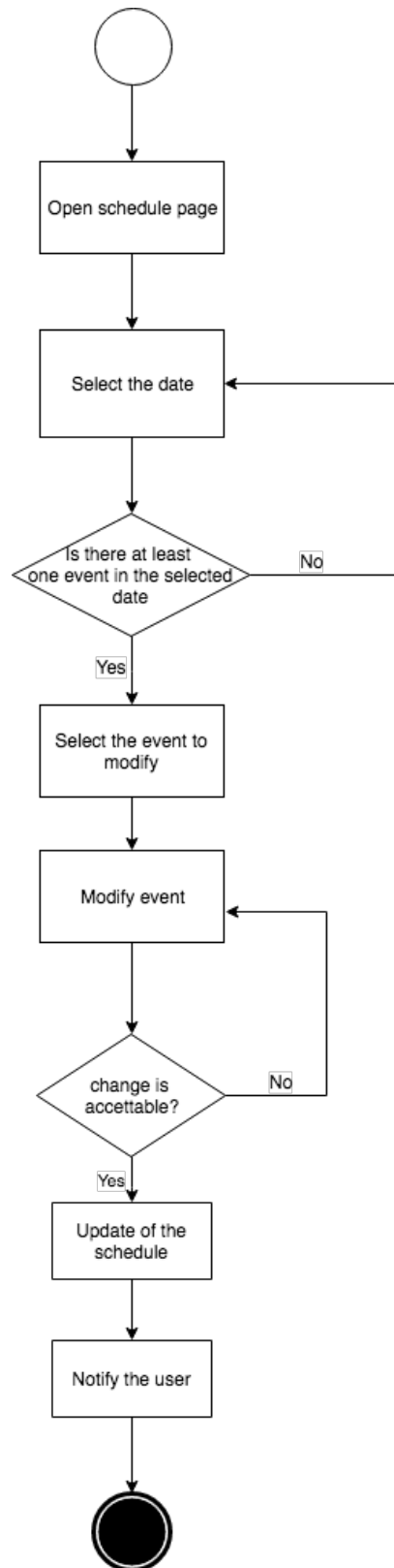


Figure 3.21: Activity Diagram : Modify an already compiled schedule for a certain day.

## Scenario 8

Mark is setting up his own preferences about the means of transport so once here he realize that is possible to set a custom time for the lunch interval and the length that he prefers for the break. He decides that for the lunch is enough thirty minutes and set the time in which he wants to have break.

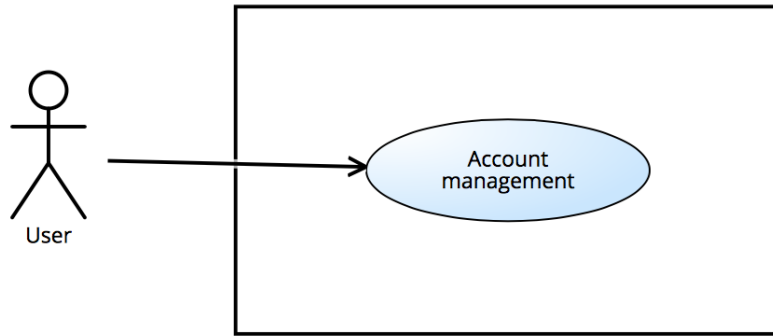


Figure 3.22: Use Case: Manage his preference about time for break

### 3 Specific Requirements

<b>Name</b>	Manage his preference about time for break
<b>Actor</b>	User
<b>Entry Condition</b>	The user is already logged in and wants to change his preferences
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user clicks on the menu button and find the page about the account management</li><li>• The system provides all the information that it know about the user and permit to modify all data defined editable</li><li>• The user inserts the desired hour that he is used to have break</li><li>• The user inserts the estimate duration of the break</li><li>• The system update the user preferences</li></ul>
<b>Exit condition</b>	The user set his preferred time slots to have a break
<b>Exceptions</b>	The user select an end time that is before the start time of the break so the system return an error. If the user inserts a break longer that the time between the start and the end of the break the system returns an error.

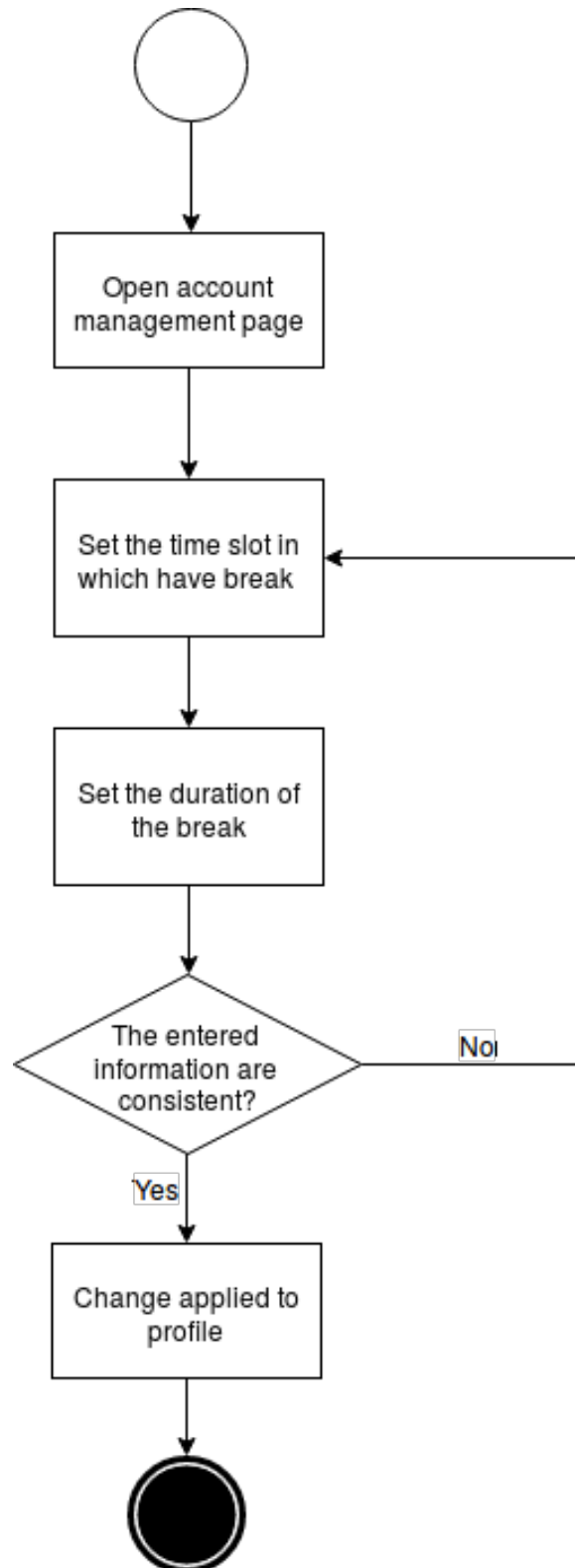


Figure 3.23: Activity Diagram : Modify an already compiled schedule for a certain day

**Scenario 9**

Mark has already finish the meeting of the 6pm and he must go to the meeting of 6,30pm, so he opens Travlendar+ application and see the best journey in order to his preferences about means of transport that can permit he to reaches the destination in the way he prefers. So Mark receive the information about the current journey and all the information about traffic jam, status of public transport service means and strikes that the third part services provide to the application.

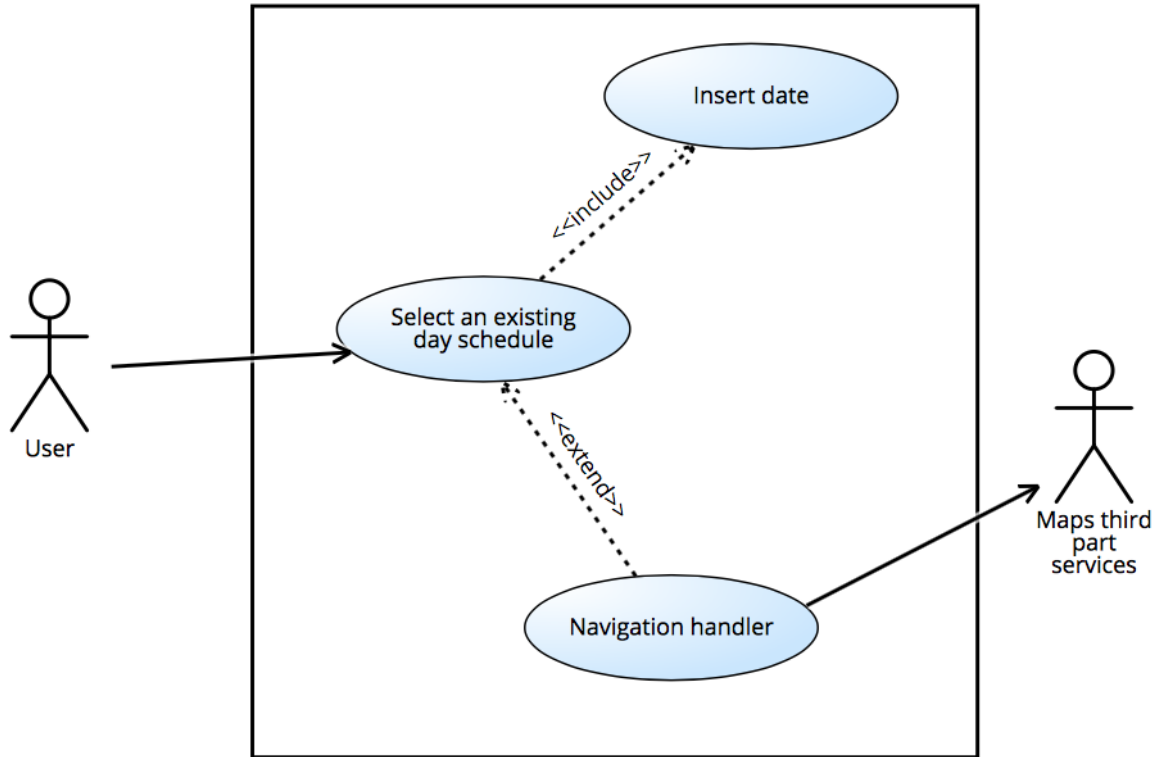


Figure 3.24: Use Case: Receive information about the current journey

### 3 Specific Requirements

<b>Name</b>	Receive information about the current journey
<b>Actor</b>	User, Maps third part services
<b>Entry Condition</b>	The user is already logged in to receive the information about the current journey.
<b>Goal</b>	9
<b>Event Flow</b>	<ul style="list-style-type: none"><li>• The user selects from the homepage the event of the current day schedule of which he wants to have the journey</li><li>• The system shows the event's information and also the map with the position of the place where the event take place</li><li>• The user clicks on the map</li><li>• The system opens the navigation system providing the journey's information</li></ul>
<b>Exit condition</b>	The user reach the destination on time for the next event
<b>Exceptions</b>	The user hasn't got any events during the current day. If the Location of the device is not available the navigator can provide an error.



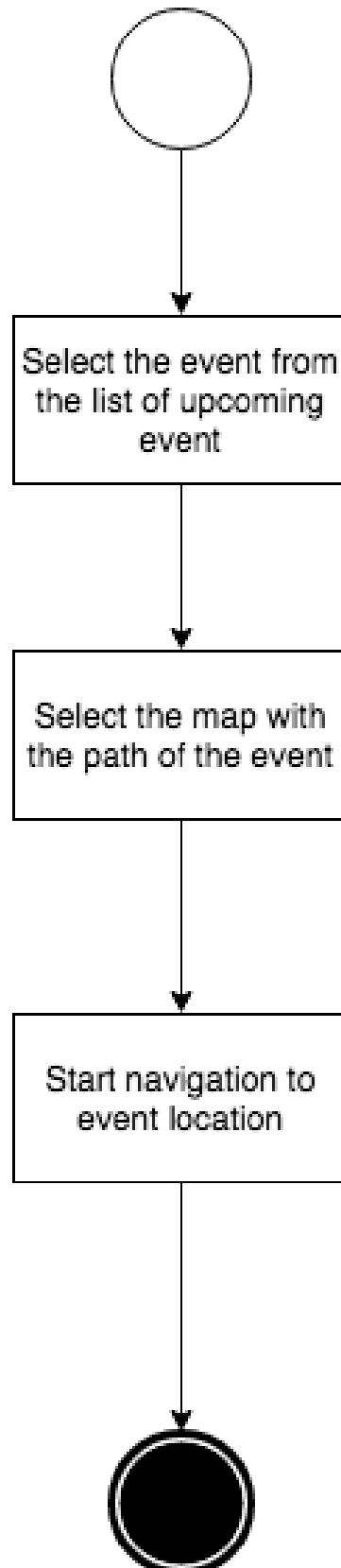


Figure 3.25: Activity Diagram : Receive information about the current journey

### 3.3 Performance Requirements

As the application will operate in large cities and therefore interacting with many users at the same time, the system must guarantee a certain reliability and fault tolerance. In particular, requests to server must be fulfilled within 5 seconds serving up to 10000 users (or 1000 parallel requests)

### 3.4 Design Constraints

The system has to memorize and store users' tickets and subscriptions in accordance with the formats provided by the local public transport services. It also has to memorize the users' driving licence with the correct format.

### 3.5 Software System Attributes

#### Availability

The system is requested to have an availability around 98% corresponding to a maximum downtime of 3.36h per week. In order to achieve this, the server used must be part of a server farm. In this way, even if a single machine ends up being unreachable, the others still work and keep the OS up.

#### Security

The system has to guarantee that only users can actually browse and access to all the application's feature, like the creation of a schedule, the purchase of tickets and so on. On the other hand, guests must be able only to check the best route to a certain destination. Moreover, all the users' informations cannot be viewed nor modified by other people. For this purpose the system, before the creation of a new event, requires a login (composed by user and password).

It is also fundamental the persistence of such datas, achieved with periodic backups which will be the base of a system recovery. Every exchange of information between server and client must be encrypted in order to avoid leaks of sensitive data.

#### Portability

The system must be developed to ease the use of the application on any mobile device. Furthermore, it has to be developed in order to run smoothly on different kinds of tablets or smartphones with several OS like Android, iOS and Windows Phone. The access to the application will be also available from any modern browser on PC.

#### **Maintainability**

The system must be implemented with an architecture that will ease the evolution of the software and its maintainability. For such purpose, the developing will require the use of an Object Oriented Language to partition the code in several classes and make it more readable. It is also required a documentation about the used method's behaviour and guarantee an easy approach to the code in future uses.

## 4 Formal Analysis using Alloy

```
open util/boolean
2 open util/integer

4 // Definition of class User
sig User{
6 username: one Username,
  name: one Name,
8 surname: one Name,
  mail: one Mail,
10 phone: one Phone,
  drivingLicense: lone DrivingLicense,
12 creditCard: one CreditCard,
  ticketList: set Ticket,
14 meansPref: Means -> Bool,
  breakPref: some Event,
16 journeyList: set Schedule,
  sharedSchedule: set Schedule
18 }

20 // Definition of class Ticket
sig Ticket{
22 ID: one ID,
  evaluDate: one Int,
24 purchaseDate: one Int,
  cost: one Int,
26 validMeans: some TypeMeans,
  type: one TypeTicket
28 }

30 // Definition of class Means
sig Means{
32 name: one MeansName,
  type: one TypeMeans,
34 status: one Bool
  }

36 // Definition of class Journey
38 sig Journey{
  startingTime: one Int,
```

```

40 duration: one Int,
   meansUsed: some Means,
42 destEvent: one Event,
   startPosition: one Position
44 }

46 // Definition of class Schedule
   sig Schedule{
48   day: one Int,
   journeyList: some Journey,
50   ID: one ID
   }

52 // Definition of class Position
54 sig Position{
   longitude: one Float,
56 latitude: one Float
   }

58 // Definition of class Weather
60 sig Weather{
   date: one Int,
62 type: one TypeWeather,
   location: one Position
64 }

66 // Definition of class Event
   sig Event{
68   name: one EventName,
   start: one Int,
70 duration: one Int,
   location: one Position,
72 type: one EventType
   }

74 // Basic definition of utility classes
76 sig Name{}
   sig Mail{}
78 sig Phone{}
   sig DrivingLicense{}
80 sig CreditCard{}
   sig ID{}
82 sig Username{}
   sig MeansName{}
84 sig EventName{}
   sig Float{}

```

```

86 // Definition of all the enum required with the matching options
87 enum EventType {MEETING, BREAK, THEREEVENT}
88 enum TypeTicket {SINGLE, SUBSCRIPTION, CARNET, OTHER}
89 enum TypeMeans {CAR, TRAIN, BUS, SUBWAY, BIKE, WALK}
90 enum TypeWeather {SUNNY, WINDY, STORMY, SNOWY, CLOUDY, RAINY}
91
92 pred show(u: User) {}
93
94 // This fact contribute to create a more readable graph while we
95 // ↪ generate the world show below
96 fact WorldOptimalView{
97     #User=2 and (all user: User | #user.journeyList=2)
98 }
99
100 // The duration of the journey must be greater or at least zero
101 fact JourneyDurationGreaterThanZero{
102     all j: Journey | j.duration ≥ 0
103 }
104
105 // The duration of the event must be greater or at least zero
106 fact EventDurationGreaterThanZero{
107     all e: Event | e.duration ≥ 0
108 }
109
110 // The start time of the event must be greater or at least zero
111 fact EventStartGreaterThanZero{
112     all e: Event | e.start ≥ 0
113 }
114
115 // The start time of the journey must be greater or at least zero
116 fact JourneyStartGreaterThanZero{
117     all j: Journey | j.startingTime ≥ 0
118 }
119
120 // The date in the weather classes must be greater or at least zero
121 fact WheatherDateGreaterThanZero{
122     all w: Weather | w.date ≥ 0
123 }
124
125 // The day which refer the schedule must be greater or at least zero
126 fact ScheduleDayGreaterThanZero{
127     all s: Schedule | s.day ≥ 0
128 }
129
130 // The purchase date of the ticket must be greater or at least zero

```

```

132 fact TicketPurchaseDateGreaterThanOrEqualToZero{
133     all t: Ticket | t.purchaseDate ≥ 0
134 }

135 // The username of the user must be distinct
136 fact UsernameDistinct{
137     no disj user1, user2: User | user1.username = user2.username
138 }

139 // The mail of the user must be distinct
140 fact MailDistinct{
141     no disj user1, user2: User | user1.mail = user2.mail
142 }

143 // The driving licence of the user must be distinct
144 fact DrivingLicenseDistinct{
145     no disj user1, user2: User | user1.drivingLicense = user2.
146         ↪ drivingLicense
147 }

148 // The schedule date of the schedule of a single user must be
149     ↪ distinct
150 fact ScheduleDateDistinct{
151     all user: User | (no disj schedule1, schedule2: user.journeyList
152         ↪ | schedule1.day = schedule2.day)
153 }

154 // The ticket ID must be distinct
155 fact TicketIdDistinct{
156     no disj ticket1, ticket2 : Ticket | ticket1.ID = ticket2.ID
157 }

158 // The cost of the ticket must be greater than zero
159 fact TicketCostPositive{
160     all ticket1 : Ticket | ticket1.cost > 0
161 }

162 // For one position in one date is possible to have only one weather
163 fact OneWeatherForPositionDate{
164     no disj w1, w2: Weather | (w1.location = w2.location and w1.date = w2
165         ↪ .date)
166 }

167 // All position must be different
168 fact PostionDistinct{

```

```

172     no disj p1,p2: Position | (p1.latitude=p2.latitude and p1.
        ↪ longitude=p2.longitude)
    }
174
    // The means that the user use must be one of the selected
176 fact RightJourneyMeans{
    all user1 : User | (user1.journeyList.journeyList.meansUsed.type
        ↪ in (user1.meansPref.boolean/True).type)
178 }

180 // All the means must be a boolean associate
fact AllMeansWithPref{
182     all m : Means, u : User | m.(u.meansPref)=boolean/True or m.(u.
        ↪ meansPref)=boolean/False
    }
184
    // Is not possible to use damaged means for the journey
186 fact NoDamageMeans{
    all user1 : User | user1.journeyList.journeyList.meansUsed.
        ↪ status=boolean/True
188 }

190 // In all journey the start and end point must be different
fact DifferentPositionStartEnd{
192     all journey1 : Journey | journey1.startPosition≠journey1.
        ↪ destEvent.location
    }
194
    // It is not possible to use the bike while rain
196 fact RainyDay{
    all journey1 : Journey, weather1 : Weather | (journey1.
        ↪ startingTime=weather1.date and weather1.type=RAINY)
        ↪ implies !(journey1.meansUsed.type=BIKE)
198 }

200 // It is not possible to associate a ticket to the travel mode "WALK
    ↪ ", "BIKE" and "CAR"
fact TicketAssociated{
202     all user1 : User | (user1.journeyList.journeyList.meansUsed.type
        ↪ ≠WALK and user1.journeyList.journeyList.meansUsed.type≠
        ↪ BIKE and user1.journeyList.journeyList.meansUsed.type≠CAR)
        ↪ implies (user1.ticketList.validMeans.type=user1.
        ↪ journeyList.journeyList.meansUsed.type and user1.
        ↪ ticketList.evalDate ≥ user1.journeyList.day)
    }
204

```



```

// in breakPref the type of the event is "BREAK"
206 fact BreakEvent{
    all user1 : User | user1.breakPref.type=BREAK
208 }

// All the schedule ID must be different
210 fact ScheduleIdDistinct{
212     no disj sched1,sched2 : Schedule | sched1.ID=sched2.ID
    }

214 // All the means name must be different
216 fact MeansNameDistinct{
    no disj m1,m2 : Means | m1.name=m2.name
218 }

220 // For all ticket the evaluation date must be greater or at least
    ↪ equal to the purchase date
fact TicketDateControl{
222     all ticket : Ticket | ticket.evalDate≥ticket.purchaseDate
    }

224 // In every schedule there is at least a break event for every
    ↪ break preference
226 fact AtLeastOneBreak{
    all user: User | (all break: user.breakPref, schedule: user.
        ↪ journeyList | (one event: schedule.journeyList.destEvent |
        ↪ event = break))
228 }

230 // Every schedule in the list of shared event of a user must be in a
    ↪ journeyList of an another user
fact SharedEventExist{
232     all user: User | (all shared: user.sharedSchedule | (one user2:
        ↪ User | (one sched: user2.journeyList|sched.ID=shared.ID)))
    }

234 // All event must be associated to a journey
236 fact EveryEventInJourney{
    all event: Event | (some journey: Journey | event in journey.
        ↪ destEvent)
238 }

240 // All journey must be associated to a schedule
fact EveryJourneyInSchedule{
242     all journey: Journey | (one sched: Schedule | journey in sched.
        ↪ journeyList)

```

```

}
244
// All schedule must be associated to a user
246 fact EveryScheduleIsOfAUser{
    all sched: Schedule | (one user: User | sched in user.
        ↪ journeyList)
248 }

250 // There is no event or journey that overlap for a schedule
fact NoEventsOverlap{
252     all user: User, schedule: user.journeyList | no disj j1, j2:
        ↪ schedule.journeyList |
        ( (j2.startingTime < ((j1.destEvent.start).plus[j1.destEvent.
            ↪ duration])) and (j2.startingTime > j1.startingTime) )
254 }

256 // Every journey start in a different time from another
fact JourneyStartDistinct{
258     all u: User | (all sh: u.journeyList | (no disj j1, j2: sh.
        ↪ journeyList | j1.startingTime=j2.startingTime))
    }
260

    // Is not possible to use the car for a journey if the field of the
    ↪ driving licence of the user is empty
262 fact NoCarWithoutDrivingLicence{
    all user: User | (user.drivingLicense = none) implies (not (CAR
        ↪ in boolean/True.(~(user.meansPref)).type ) )
264 }

266 // The journey must finish at the starting time of the event
fact JourneyDuration{
268     all journey: Journey | ( ((journey.startingTime).plus[journey.
        ↪ duration]) = journey.destEvent.start)
    }
270

    // All ticket must be valid for only public means
272 fact TicketOnlyForPublicMeans{
    all ticket : Ticket | ( (not (WALK in ticket.validMeans) ) and (
        ↪ not (BIKE in ticket.validMeans) ) and (not (CAR in ticket.
        ↪ validMeans) ) )
274 }

276 // Assert that checks there is no event or journey that overlap for
    ↪ a schedule
assert NoEventsOverlap{

```

```

278   all user: User, schedule: user.journeyList | no disj j1, j2:
      ↪ schedule.journeyList |
      ( (j2.startingTime < ((j1.destEvent.start).plus[j1.destEvent.
      ↪ duration])) and (j2.startingTime > j1.startingTime) )
280 }

282 // Assert which checks that in every schedule there is at least a
      ↪ break event for every break preference
assert AtLeastOneBreak{
284   all user: User | (all break: user.breakPref, schedule: user.
      ↪ journeyList | (one event: schedule.journeyList.destEvent |
      ↪ event = break))
    }

286 //Assert which checks that every schedule in the list of shared
      ↪ event of a user must be in a journeyList of an another user
288 assert SharedEventExist{
      all user: User | (all shared: user.sharedSchedule | (one user2:
      ↪ User | (one sched: user2.journeyList | sched.ID=shared.ID)))
290 }


292 run show for 4 but 5 int

294 check NoEventsOverlap
check SharedEventExist
296 check AtLeastOneBreak

```

**Executing "Run show for 4 but 5 int"**

Solver=sat4j Bitwidth=5 MaxSeq=4 SkolemDepth=1 Symmetry=20  
36334 vars. 1752 primary vars. 101041 clauses. 246ms.

 **Instance** found. Predicate is consistent. 258ms.

**Executing "Check NoEventsOverlap"**

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20  
13956 vars. 813 primary vars. 34593 clauses. 79ms.

No counterexample found. Assertion may be valid. 6ms.

**Executing "Check SharedEventExist"**

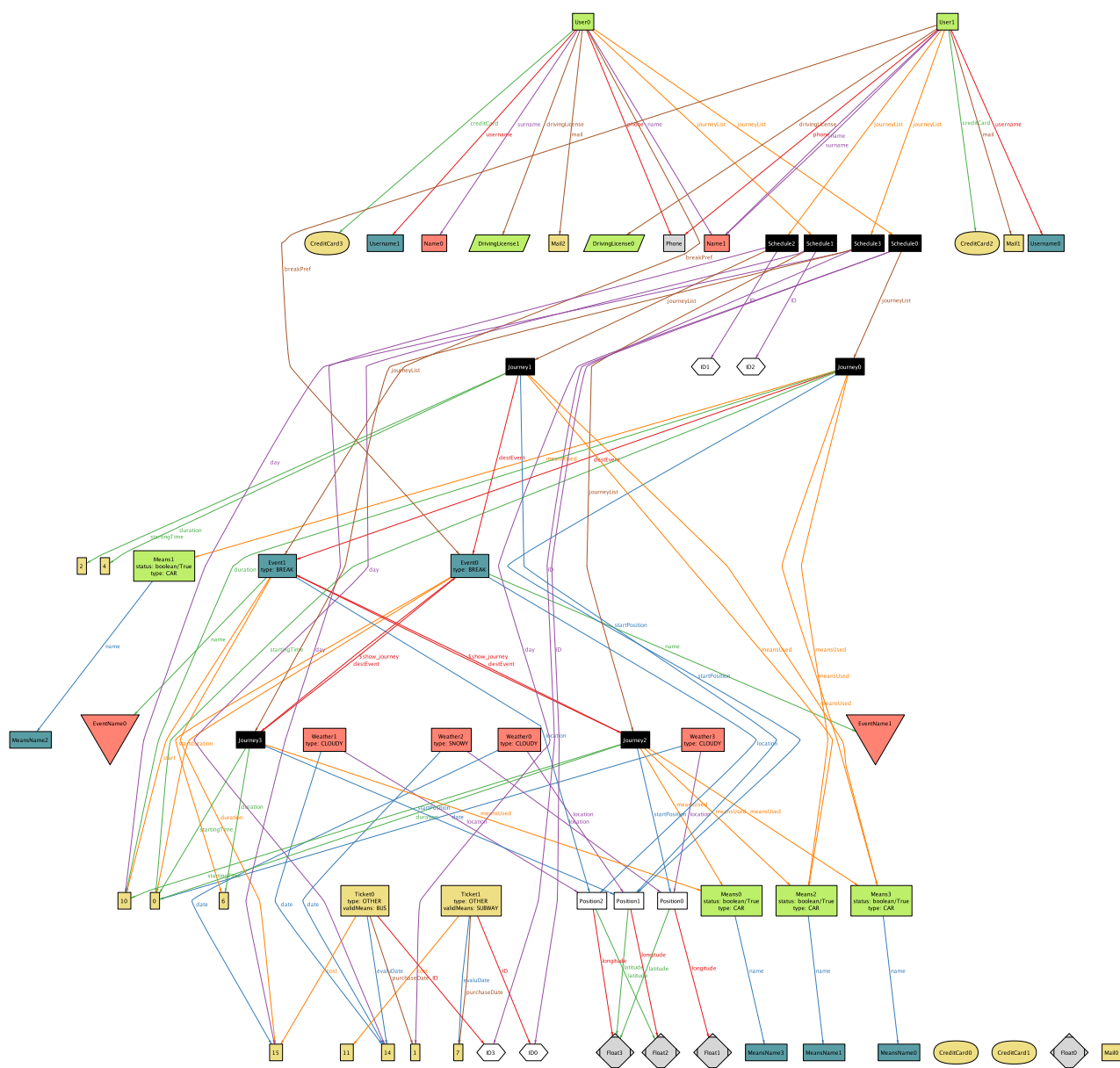
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20  
13111 vars. 807 primary vars. 32024 clauses. 72ms.

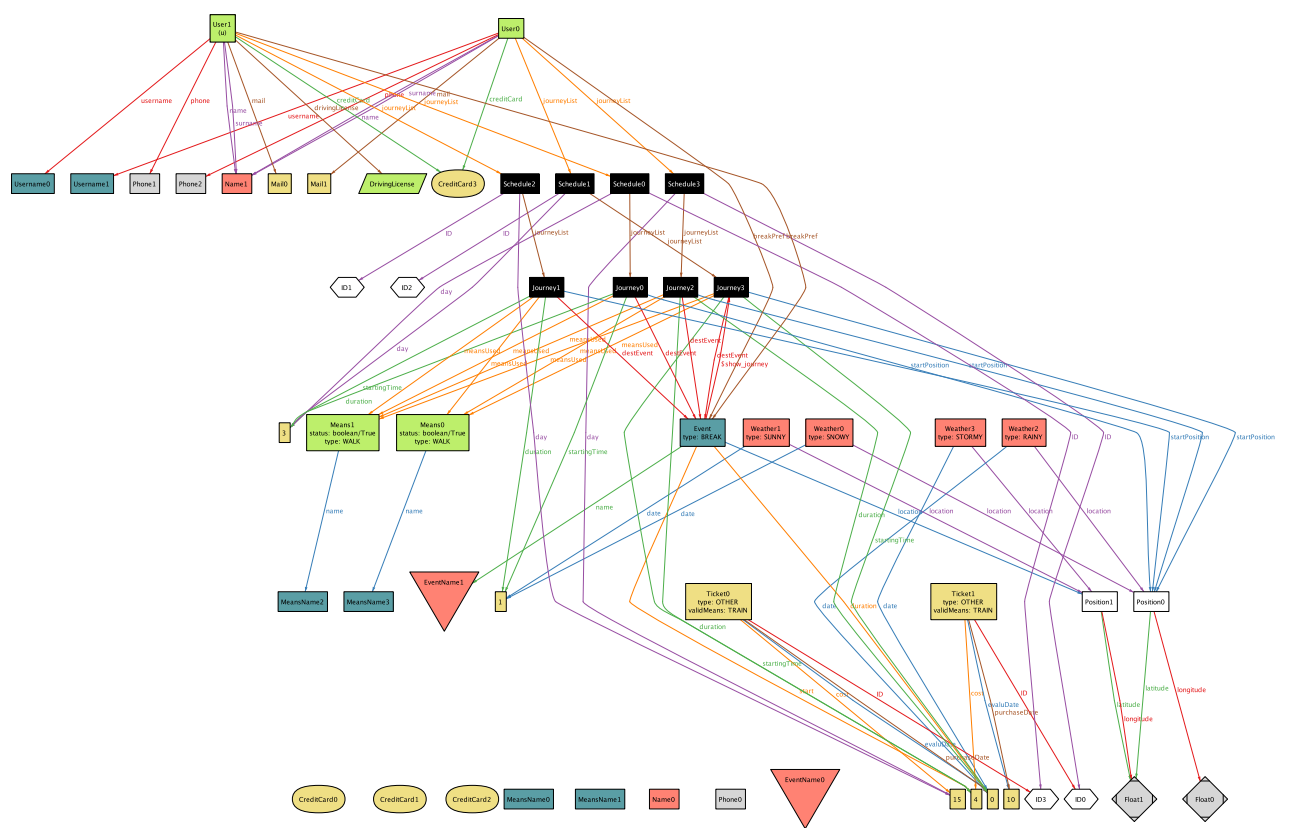
No counterexample found. Assertion may be valid. 4ms.

**Executing "Check AtLeastOneBreak"**

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20  
13110 vars. 810 primary vars. 31971 clauses. 77ms.

No counterexample found. Assertion may be valid. 3ms.





## 5 Effort Spent

While working on the project, we always met to discuss main topics to provide more consistence to the project:

Name	Effort
Lukasz Moskwa	Group 21h and 8h alone
Marco Mussi	Group 21h and 8h alone
Gianluigi Oliva	Group 21h and 8h alone

## 6 References

- **Documentation about availability:**  
[https://en.wikipedia.org/wiki/High\\_availability](https://en.wikipedia.org/wiki/High_availability)
- **Documentation about alloy:**  
<http://alloy.mit.edu/alloy/>
- **Thesis about alloy:**  
[http://computerscience.unicam.it/culmone/?download=Tesi\\_Claudio\\_Vallorani%28066729%29.pdf](http://computerscience.unicam.it/culmone/?download=Tesi_Claudio_Vallorani%28066729%29.pdf)
- The original Travlendar application:  
<http://score-contest.org/2018/projects/travlendar.php>
- The revised document of the assignment:  
<https://goo.gl/9m1ojy>
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- IEEE Std 1016tm-2009 Standard for Information Tecnology-System Design-Software Design Descriptions.