Politecnico di Milano
A.A. 2017-2018
Software Engineering II project

Travlendar+

Requirements **A**nalysis
and
Specifications **D**ocument

Mirko Mantovani (893784), Matteo Marziali (893904)

October 28, 2017

Contents

1 Introduction

1.1 Purpose

The main purpose of Travlendar+ is to create a software that allows users to easily manage their daily meetings and commitments, by providing some useful features such as finding the best means of transport to reach the appointment place and easily know the quickest route available to be punctual. Specifically, we want to realize a product which is able to:

- Provide a calendar and the possibility to memorize events and appointments on it.
- Automatically compute and account for travel time between appointments to make sure that the user is not late for them.
- Automatically generate warnings to notify the user that at least two meetings are overlapping.
- Provide routes and travels according to user preferences about the preferred/prohibited travel means and daily breaks set.
- Provide the possibility to add reminders for a meeting in order to prevent the users forgetting their appointments.
- Allow the users to modify appointment schedules.
- Automatically notify people involved in a specific meeting if the user is late and has selected this feature previously.

On the other hand, the purpose of this paper is to define in a detailed way all the functions and requirements of our application. In doing this, we start focusing on a brief overview to characterize the product with relevance to its interaction with the world, then we will proceed deeply in analysing which functions are relevant and should be provided, and which requirements are needed to the stakeholders.

1.2 Scope

Even if we are very confident about the success of our idea, initially, Travlendar+ will have a restricted domain, indeed we will experiment it only in the Italian city of Milan. In order to provide the most complete assistance, Travlender+ will suggest different paths and a wide range of transports such as bike, even shared, your car or a shared one, taxi, bus, and also.. your feet! It goes without saying that to organize this kind of service in the most valuable way we must interact with a huge number of local institutions which provide different services in town.

1.2.1 The world and the machine

The first model of the system to be presented is the model "The world and the machine" by M. Jackson and P. Zave. This model highlights the division between phenomena that happen entirely either in the world or in the machine, and those that are shared between the two of them.

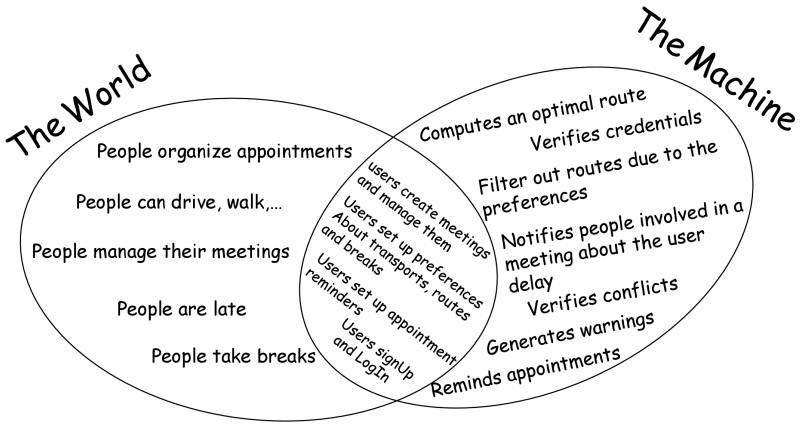

Figure 1: The world-machine chart.

1.3 Goals

G1 Memorizing and organizing both events and appointments on the calendar.

G2 Being sure to reach the appointment location in time avoiding delays.

G3 Being sure not to schedule overlapping meetings.

G4 Being able to use only travel means that fit with user preferences.

G5 Preventing the user to forget an appointment.

G6 Allowing the users to modify appointment schedules.

G7 Notifying other people involved in a meeting about user eventual delay.

1.4 Definition and Acronyms

1.4.1 Definitions

- **App:** this is the abbreviation for application, in particular this term is used meaning a mobile application.
- Delay notification function: this phrase refers to the function which allows to notify the participants of a meeting through an email in case the user is late.
- **Travel:** a travel is any suggested path that goes from the starting point to the meeting location.
- **Route:** this term is used as a synonym of travel.
- **Warning:** warning is the word used to define the conflict between two meetings.
- **Conflict:** a conflict between two or more meetings is what enables the creation of a warning, it means that the set of meetings in conflict are scheduled too close in time in order for the user to be able to attend them all in time.
- **Calendar:** the calendar contains the list of meetings and is grouped by day.
- **Meeting:** is an important keyword of the application, it includes all the informations of an appointment.
- **Reminder:** a reminder is a sort of an alarm triggered at a certain time before an appointment is starting.

1.4.2 Acronyms

- ETA: estimated time of arrival: the time, estimated by the system, that the closest available taxi will take to get to the starting location of the ride.
- API: application programming interface; it is a set of routines, protocols, and tools for building software applications on top of this one.

1.5 Revision

v1.1 after project class meeting on 11/10/17 Added lists to goal, functions and assumptions; Revised section 1 and 2

1.6 Actors

- *Guest*:
- *Logged in users*:

1.7 References

- The document with the assignment for the project
- The IEEE Standard for SRS

1.8 Document Structure

This document is structured in three parts:

- Introduction: In the first introductory section, we give a short description both of the goals and of the environment which our app has to deal with. Moreover, we explain some notes useful to understand and read the whole paper.
- Overall Description: gives a general description of the application, focusing on the context of the system, going in details about domain assumptions and constraints. The aim of this section is to provide a context to the whole project and show its integration with the real world and showing the possible interactions between the user, the system and the world itself.
- Specific Requirements: this section contains all of the software requirements to a level of detail aimed to be enough to design a system to satisfy said requirements, and testers to test that the system actually satisfies them. It also contains the detailed description of the possible interactions between the system and the world with a simulation and preview of the expected response of the system with given stimulation.

Finally, we express the requirements through the Alloy model, which allows us to define the interactions, the functions and the constraints that characterize Travlendar+ using a formal language. The document ends with a short note about the effort spent in producing it and at last you can also find useful references.

2 Overall description

2.1 Product perspective

Our idea is to create a personal companion application to help users managing and organizing their daily life. According to this intention, we would like to realize an extremely friendly user interface and a lightweight software in order to make Travlendar+ affordable to many people and runnable by many devices. In order to make use of every functionality the devices require GPS service and an internet connections for most of the services. Since Travlendar+ is going to offer many routes depending on different travel means, it will necessarily have to interact with many institutions such as public transport and car/bike sharing providers. This aspect will affect both the software and the hardware design. Indeed, it is necessary to query data about the shared cars, bikes and the taxis location around the city and to retrieve information about trains and buses schedules. Hence, our system must be very fast and dynamic to support a huge number of query in a few seconds, moreover to interview external databases it's strictly required that the users have an active internet connection. Concerning the hardware, we intend to have a database which only contains username and password of all our customers. For sure, it seems useless having a database for those kinds of data but in our idea this choice allows the app to be updated and improved easily in the future, for instance saving on the database clients' routes and meetings.

2.2 Product functionalities

F1 Signup and Login: Travlendar+ users must sign up the first time they intend to create a meeting and further usages of the app will require a login to access all its functionalities.

F2 Meeting creation: This is the most important function of the app, it allows to generate an event related to an appointment. It requires the user to define all the details such as date, time, location, starting point, preferences etc.

F3 Preferences set up: An important feature of Travlendar+ consists in allowing the user to filter out specific routes depending on some constraints about the travel, or to set break-dedicated time slots.

F1 Warnings management: In case the user seems to be late, the app generates a warning to alert him. Hence, he can either ignore it or postpone the appointment.

F4 Delays management: If the app had noticed, according to the estimated travel time, that the user is in late, and he previously had inserted the email address of the meetings participants, Travlendar+ would notify them about the delay.

F5 Route generation: the main hidden function of Travlendar+ is to automatically compute and suggest to the user the best travel among those which fit the preferences he has selected.

F6 Reminder management: The applications allows users to set up reminders for a certain Meeting in order for the user not to be late for it.

F7 Recurrent events management: The smartest function Travlendar+ will offer; it consists in allowing the user to select events to be rescheduled periodically just creating one meeting. Done this choice, the app. Automatically manages to reschedule the specific meeting according to the period that the user establish, for instance one week, one month.

F8 Update meetings: This function is both basic and relevant, it allows the user to customize his meetings after their creation. In other words, through this functionality the user can modify each one meeting details, even in case of a warning is generated.

2.3 User characteristics

According to our idea, Travlendar+ does not have a specific customers range, it is supposed to be used by both male and female, whatever their age is. Obviously, considering that we intend to produce a mobile application, people who want to use Travlendar+ should be familiar with a portable device like a smartphone or a tablet. For sure, users should respect several requirements due to

the travel means they are going to take. For instance, when cars are selected as active in the means of transport list, the user is supposed to have a valid driver licence, taking the bus is allowed only with an appropriate ticket. Moreover, users interested in dealing with Travlendar+ services must have an e-mail address, primarily due to register and authenticate themselves, secondly to use the delay notification function.

2.4 Assumptions and dependencies

2.4.1 Domain Assumptions

- The user is supposed to attend every meeting
- A meeting should not take more time than expected.
- If a user has a meeting in a specific location, he's supposed to be there at the end of the meeting, and the app will compute a route based on that information
- The Metros and buses are supposed to be on time, travel times are calculated based on the expected route duration.

2.4.2 General Assumptions

A1 Signup and Login

Considering that the assignments provided do not say much generally about users without any reference to a possible signup or login, we assume that the registration is mandatory to create the first meeting, then every access to the app requires the login to manage each event saved. Please note that login parameters could be memorized to save and recover easily a user instance.

A2 Meeting management

According to the requirements, we want to develop a system which allows the user to set his preferences with regards to the travels. Moreover, we decide that he can also cancel or anticipate/postpose an event, assuming a previous reschedule agreement among the participants. It goes without saying that an appointment can also be modified, this means that a user can change either the starting location or the arrival location, the hour, the date and the other details chosen during the creation of the meeting, always making the same rescheduling agreement assumption.

A3 Warnings

Our assumptions about the warning are the following: when the system generates a warning, the app allows the user to modify the related event that could be cancelled or delayed. In case of the user postposes the meeting, if he provided the email addresses of the other people involved

in the appointment, Travlendar+ automatically will notify them that a change occurs.

A4 Routes

Concerning the routes, we decided to manage them in this way. The system generates different routes according to the user preferences, it will be the user itself to decide which itinerary fits better with him among the alternatives.

A5 Preferences

As far as the preferences are concerned, we decided that they belong to a user instead of a meeting. This means that a user cannot define different preferences for each meeting, while they are valid for every appointment.

2.4.3 Constraints

C1 Logging in whitout being signed up is prevented.

C2 Logging in, being already logged in, is impossible.

C3 Signing up, being already signed up, is impossible.

C4 Creating a meeting with the same schedule of an existent one is not allowed.

C5 Scheduling a meeting during break pauses, setted in preferences, is prevented.

C6 it is impossible to generate an appointment in the past or in an invalid date.

C7 Meeting creation requires name,date,hour,location and a starting point to be defined properly.

C8 At least one mean of travel must be selected.

C9 Every lunch break must last at least 30 minutes.

C10 If rain or snow are in the forecast, travel by bus is preferred.

C11 In case of strikes, routes involving the related travel mean are not considered.

C12 Reminders must be setted before the scheduled time of the related events.

C13 At least one travel mean must be activated in preferences.

C14 To use the 'Delay notification function' the user have to insert the email of people involved in the meeting.

3 Specific requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

Concerning the user interface requirements, we established to directly provide information about the application screens and layout through several mockups. We designed the sketch of the main pages of Travlendar+ following our aim to make a light and user friendly product but, at the same time, we have pursued an attractive and impressive style. Apparently, the coding phase could affect our desing. Hence, these sketches are not definitive, their aim is to give an idea of the application design. For this reason, if we notice that some changes are necessary due to either app. improvement or obstacles in realization, we would be ready to modify them.

Figure 2: Travlendar+ homepage.

To pursue our aim of realizing a user-friendly and light app, we decided to provide the main functions directly in the homepage. Indeed, from this page, a guest can reach the sign up form, a user can log himself in,can insert a meeting by tapping onto a day in the calendar,can manage his preferences and even access the warnings.

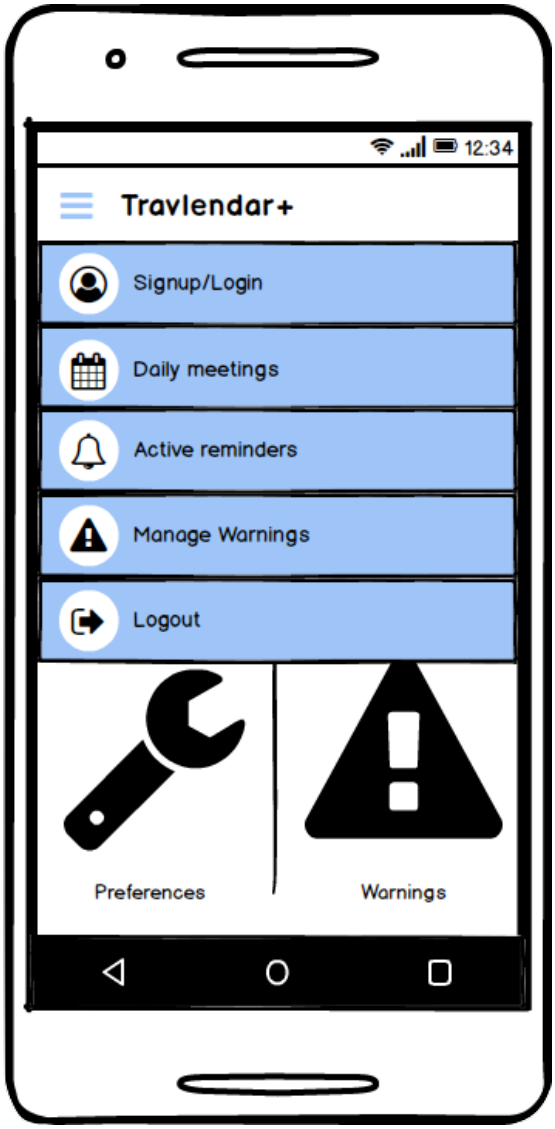

Figure 3: Quick access menu.

We thought about a quick menu to collect some secondary functions, to make them easily reachable. By tapping on the left high corner of the app. people has the opportunity to either register or log in themselves, to view only the meetings of the current day, to access the reminders they setted up, to manage the warnings and to logout themselves if they are already logged in.

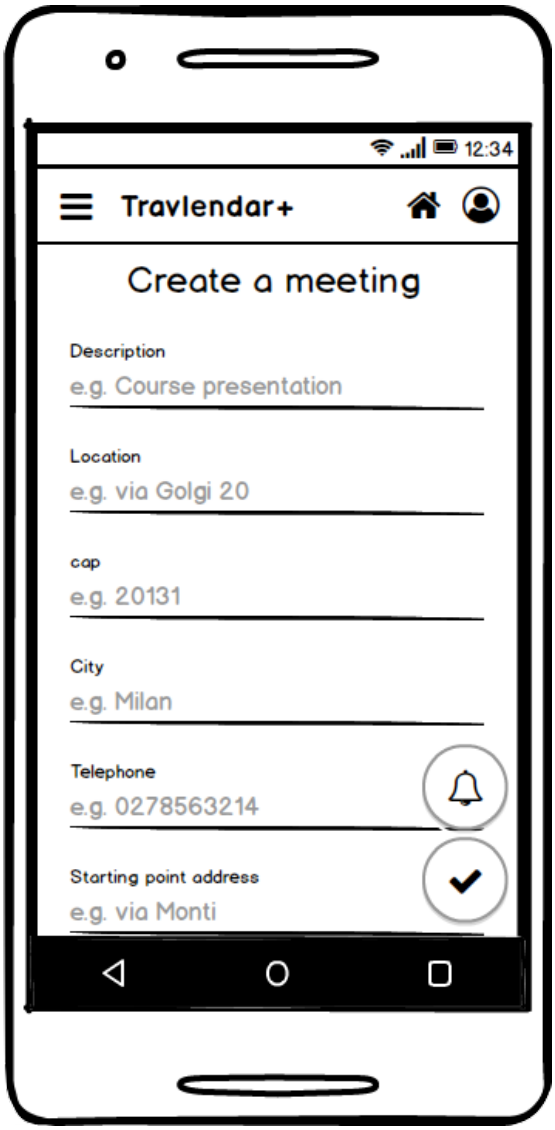

Figure 4: Meeting creation page.

This page is a very simple form, which allows the user to finalize the creation of a meeting by filling it in all its fields. The house-logo in the right corner,on the top band and near the application name, represent the return-to-homepage icon.

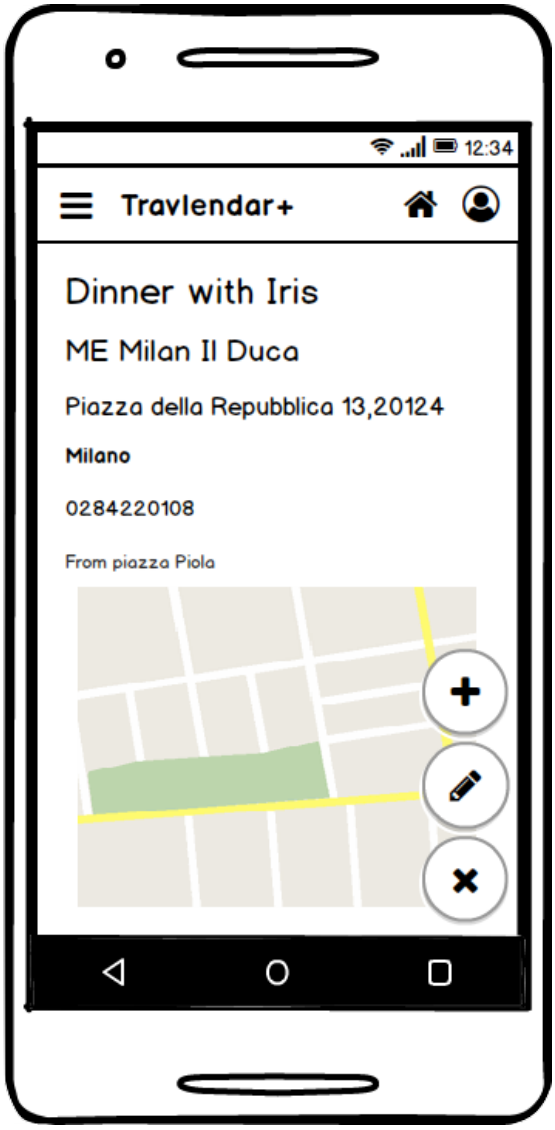

Figure 5: Meeting view page.

This screen wants to provide all the useful information related to a meeting already registered in the system. First details provided, in the highest portion of the page, regard the meeting location and the route to reach the appointment, further information are located below. In addition, on the right part of the screen, there are quick access buttons: the "plus" icon allows the user to add a reminder for the event, the "pencil" icon is to change meeting details and the "x" button provides deletion function.

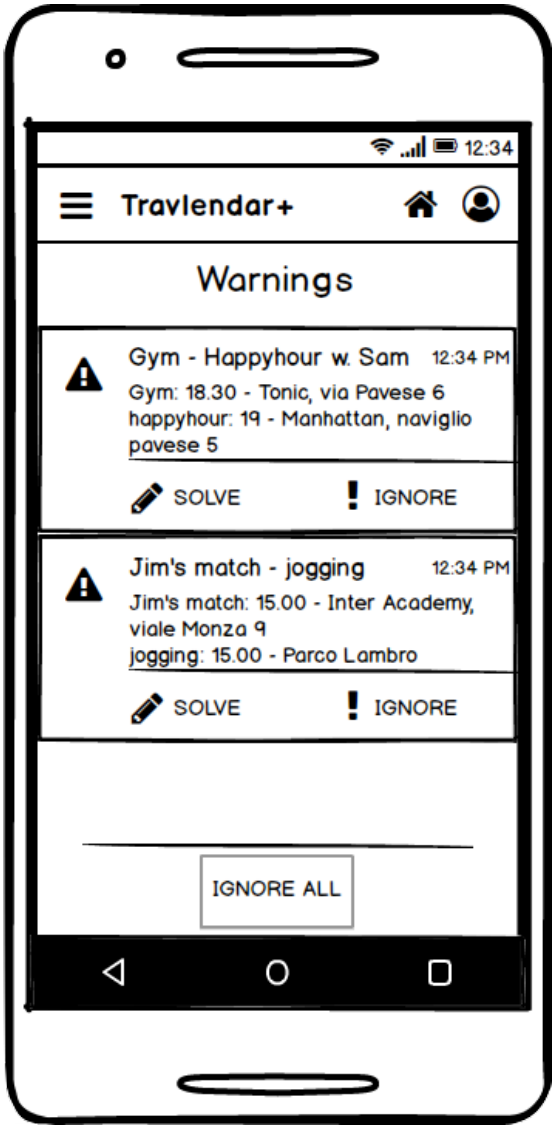

Figure 6: Warnings page

The warnings page has the role to summarize and notify all the conflict among meetings which involv the user's appointments. Every warning is represent by a dialog which points out the meetings that generate the conflict and which has two buttons, one to ignore the warning and other one to solve it by modifying the conflictual meetings. To prevent too much user's clicks, is provided an "ignore all" button, which is equivalent to tap "ignore" for each warning in the list.

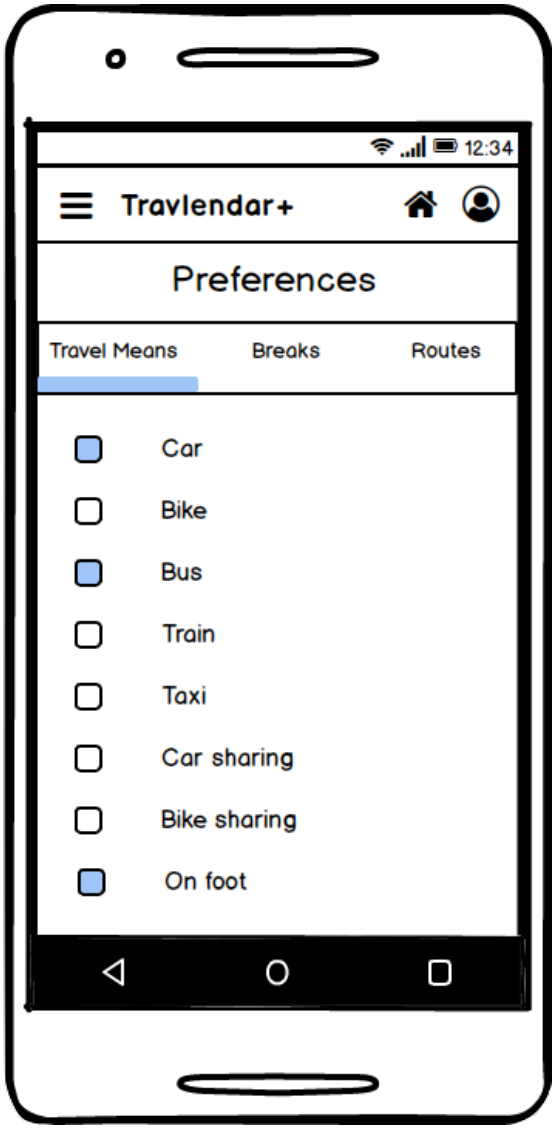

Figure 7: Travel mean preferences page.

This is a very simple page with a minimalistic design, not to uselessly load the application. However this basical view provides all the function which the user needs to select his preferences on the transports for his travels. Please note that the preference section can be switched by tapping on an specific topic just below the "Preferences" band.

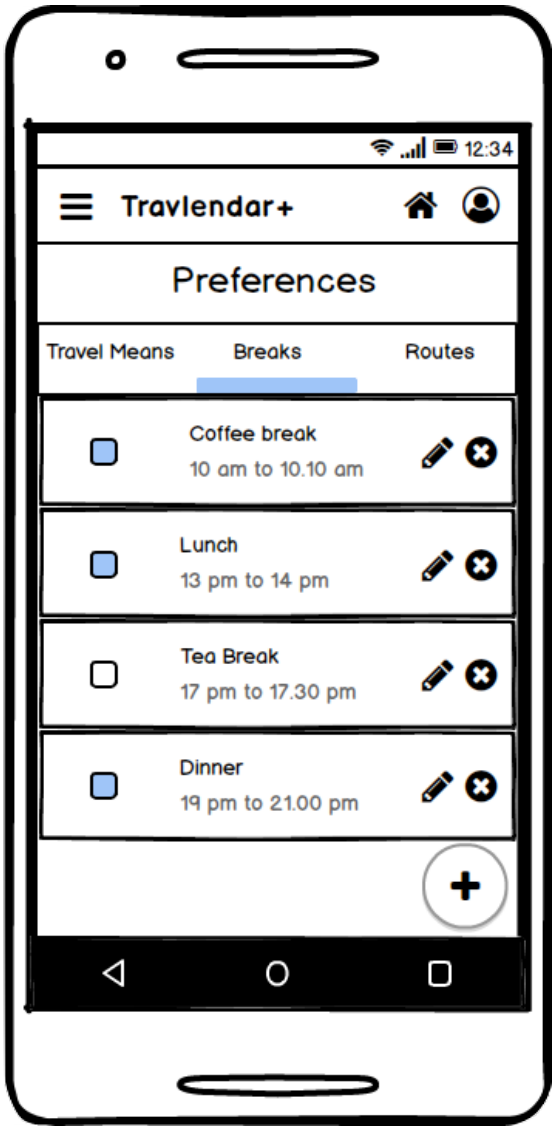

Figure 8: Break preferences page.

The breaks page consists in a list of events, that represents pauses, organized in order of generation and labeled by the type of the break. For every break are provided both the starting and the ending time, a "pencil" button to allow modifications and a "x" button to remove it. In addition, thanks to the fact that we decided to make the breaks general, that means not related to a specific day, the user has the faculty to either pick or unpick them to activate or deactivate them.

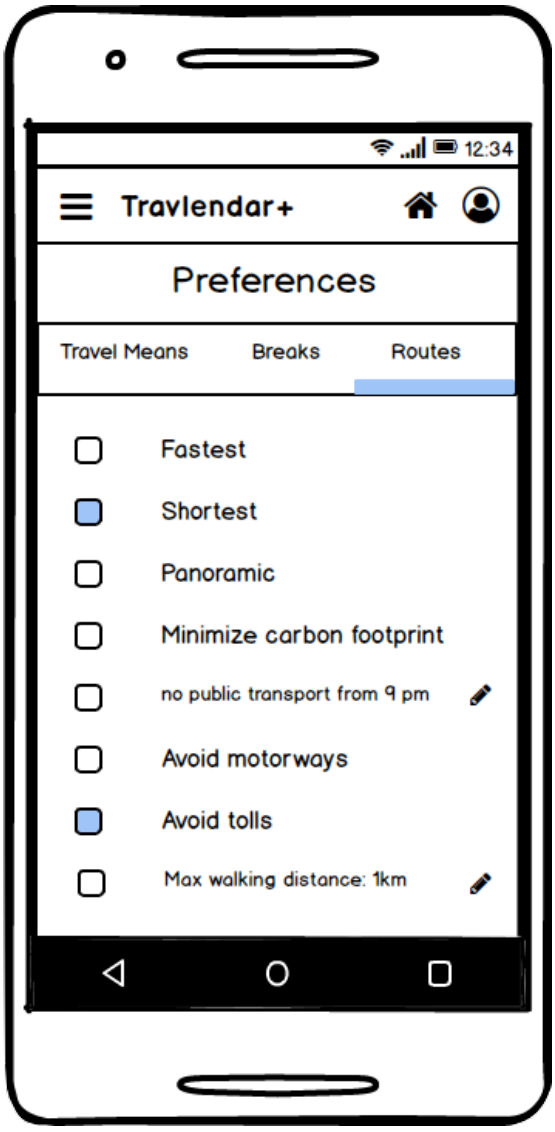

Figure 9: Route preferences page

The route preferences page is very similar to the Travel mean preferences page. The style is the same and the opportunity to pick or unpick elements too, however there are differences. Some of the preferences in this section are mutal exclusive, so the user is prevented to select more than one of them (for instance a user can select either the shortest route or the fastest). Moreover, some selection element has customizable details, they are represented with a pencil logo on the right, and can be managed by the user to best fit his preferences (for example the user can select after which hour he do not want to use public transports).

3.1.2 Software Interfaces

Considering the domain of our application, we have decided to integrate in our project some software components to create an easier and more powerful product.
In order to provide an excellent navigation service, we thought to adopt the Google Maps API, to retrieve the user location through the Google Maps' servers and databases.
In addition, we have noticed that the most complex computations which Travlendar should effort are those related to calcuating the best route. This idea suggested us to use the API of several travel mean sharing services, such as Mobike or CarToGo, to support this crucial phase. We are sure that these APIs would have allowed us to query the databases and to retrive precise information in the quickest and easiest way.
The same reasoning is applicable to forecast, needed to avoid certain routes in case of particular wheater, indeed wheater information from a specific provider are supposed to be rietrived through its APIs.
Finally, APIs of public transport agencies are required, to retrieve real time information about buses,trains and taxis and to mantain the app. up to date with relevance to all the related news, for instance about strikes.

3.2 Non functional requirements

- Simple User Interface: The user interface has to be as simple and intuitive as possible, the application should allow an average user to set up an account and start using the application understanding its functionality in no more than a dozen minutes.

- Portability. The client has to be compatible to all the major hardware and software platform on the market, this is accomplished using the web application solution presented early.

- Performance. The application should be able to calculate shortest paths very quickly in order to let the user choose the one that better fits his needs right after setting up the meeting.

- Reliability. The system should be able to guarantee the service independently of the time, 24/24, 7/7. Thus, the used services should be always available, however, since this is not a critical application, brief unavailability could be acceptable.

- Data integrity, consistency and availability. System data have to be always accessible. Hence the system should always provide a reliable access to them in normal condition. They also have to be duplicate in order to avoid data losses in case of system fault.

- Security. Hashed password should be stored in the database in order to guarantee a high level of privacy to the users. Sensible data such

as meetings details will probably be stored locally on the user device, a possible encryption might be considered but it's not a priority

3.3 Scenarios

3.3.1 Scenario One

	Creation of an event and Set up of Preferences
Actors	Jon Snow: habitual user
Flow of events	As soon as Jon receives an email from the other members of the Night's Watch, he doesn't wait and, as always, he opens Travlendar+ in order to set up the scheduled the important Meeting for saturday. He logs into the application inserting his credentials (see use case Table 4) and then from the Homepage he starts creating the new Meeting (see use case Table 6). The application suggests an optimal route using the car, as it is the fastest way to get to The Wall from Winterfell where he is currently staying. Unfortunately, Jon doesn't own a car, nor does he know what a car is, in fact he knows nothing. He deletes the Meeting straightaway (see use case **??**) and navigates through the app reaching the "Preferences set up" page at once. Here he deactivates all impossible means of transport (see use case Table 9) and leaves as active only On Foot and By Horse options. He then recreates the exact same Meeting as before and this time the system comes up with a feasible travel. Satisfied, Jon closes the app and goes on hunting rabbits.

Table 1: Scenario 1

3.3.2 Scenario Two

	Warning Solving and Reminder Addition
Actors	Jon Snow: habitual user
Flow of events	The following day, a raven from Dragonstone arrives at Winterfell, Jon's heart starts beating faster as soon as he notices the Targaryen sigil. It's Danaerys who is asking for a date with him on saturday night. Jon, unable to decide whether he can make it for both the appointments on time, lets his favourite companion decide it for him. He starts up Travlendar+ and creates the meeting (see use case Table 6). Unfortunately, the ride from Dragonstone to The Wall is long, and a warning appears on Jon's app stating that he is not going to be able to make it on time for both the meetings and a reschedule proposal is made. Jon Solves the conflict (see use case Table 8) and the warning disappears. Since he was longing for a date with Dany, Jon decides to add a Reminder (see use case Table 7) to the event in order for him to have enough time to suit up for the special date.

Table 2: Scenario 1

3.4 Use Cases

This section contains all the use cases initially described with the use cases UML model, then the most important Use Case have their own table which provide further details such as: involved actors, entry conditions, flow of events, exit conditions and exceptional conditions.

3.4.1 User Page use cases

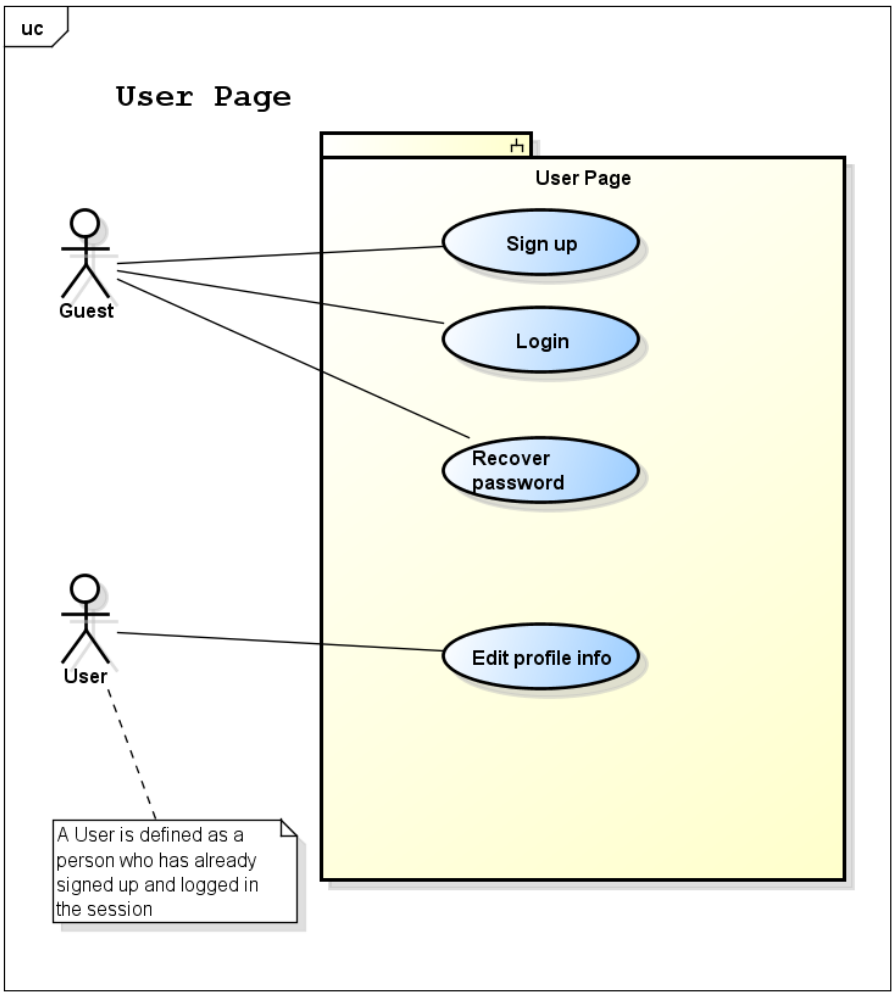

Figure 10: Use cases relative to the user registration and authentication

3.4.2 Sign up

Name	Sign up
Actors	Guest
Entry conditions	None
Flow of events	• The guest reaches the registration page containing the relative form • The guest fills up the form and clicks on "Sign up" to complete the process • The system redirects the user to his profile page and sends a confirmation email.
Exit conditions	The guest has successfully registered in the system.
Exceptions	The guest left an empty field or typed something wrong an error message is displayed and the user is asked to fill the form again.

Table 3: Sign up Use Case table

3.4.3 Login

Name	Login
Actors	User
Entry conditions	The user has already registered.
Flow of events	• The user reaches the login page containing the relative form • The user types the username and password in the login form and click on "Login" button. • The system redirects the user to the application homepage.
Exit conditions	The user has access to the application functionalities.
Exceptions	Username and password didn't correspond or the username didn't exist ,an error message is displayed and the user is asked to fill the login form again.

Table 4: Login Use Case table

3.4.4 Password Recovery

Name	Recover Password
Actors	User
Entry conditions	The user has already registered.
Flow of events	• The user reaches the login page containing the relative form • The user clicks on "Password recovery" button and is redirected to the password recovery page. • The user inserts his email and clicks on "reset password". • The system sends an email to the user with a link and instruction to reset the password. • The user chooses and types a new password and confirms. • The system redirects the user to the login page.
Exit conditions	The user has changed his password
Exceptions	The inserted email doesn't match any user in the database, it is displayed an error message and the user is asked to retype a valid email.

Table 5: Recover password Use Case table

3.4.5 Schedule Management use cases

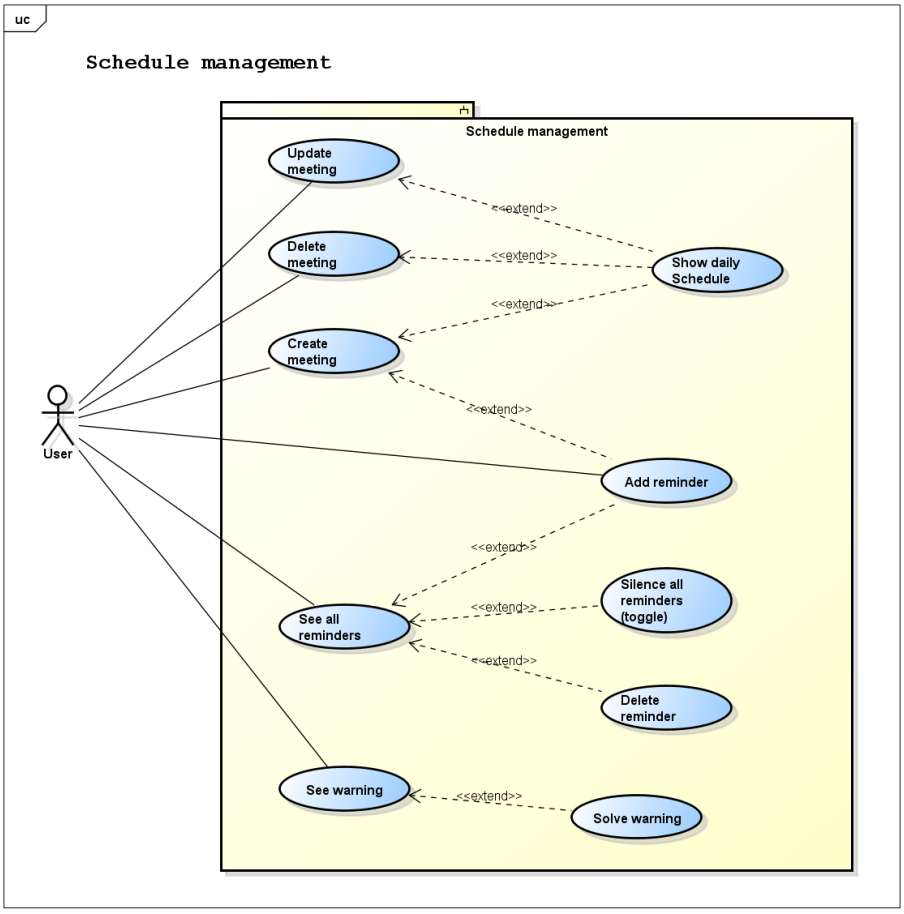

Figure 11: Main use cases showing the functionalities of Travlendar+ application relative to the meetings creation and management

3.4.6 Meeting Creation

Name	Creation of a meeting
Actors	User
Entry conditions	The user is logged in and is in the main page.
Flow of events	• The user clicks on "Create meeting" button and he is redirected to the page with the input form to create a meeting. • The user fills up the form with the meeting information (title, location, description, date and time, flags, etc...). • The system checks whether the meeting at the specified time and date is possible according to the user preferences and the current daily schedule, the optimal route is proposed. • The user is then redirected to the main page.
Exit conditions	The new meeting with the calculated optimal route is added to the user Calendar. In the case of the incompatibility of a meeting with others a warning is created.
Exceptions	The information inserted is wrong or some information is missing: a corresponding error is displayed and the user is asked to modify the inserted information accordingly.

Table 6: Meeting Creation Use Case table

3.4.7 Reminder Addition

Name	Addition of a reminder
Actors	User
Entry conditions	The user is logged in and is on the page of a meeting
Flow of events	• The user clicks on "Add reminder" button and he is redirected to the page with the input form to add a reminder. • The user fills up the form with the type of reminder and the time he wants to be reminded of the upcoming meeting. • The system adds the reminder and the user is redirected to the relative meeting page.
Exit conditions	The reminder is added to the meeting
Exceptions	There exists already an identical reminder and it is not added to the meeting

Table 7: Reminder Addition Use Case table

3.4.8 Warning Solving

Name	Solving a warning
Actors	User
Entry conditions	The user is logged in and is in the page of a warning
Flow of events	• The user clicks on "Solve warning" button and he is redirected to a page that lets him choose how to solve the conflict: the timing of two overlapping meetings can be changed, or one of the two meetings has to be canceled; • The user solves the conflict the way he wants and clicks on the button "Done". • The system checks whether the conflict has been solved and the user is redirected to the warnings page.
Exit conditions	The warning has been solved and is deleted from the system and from the list of warnings in the corresponding page
Exceptions	The warning was not solved after the user's modifications, the unresolved warning will still be present and a message stating that the conflict wasn't successfully solved is displayed. The user is redirected to the warning page.

Table 8: Warning solving Use Case table

3.4.9 User Preferences use cases

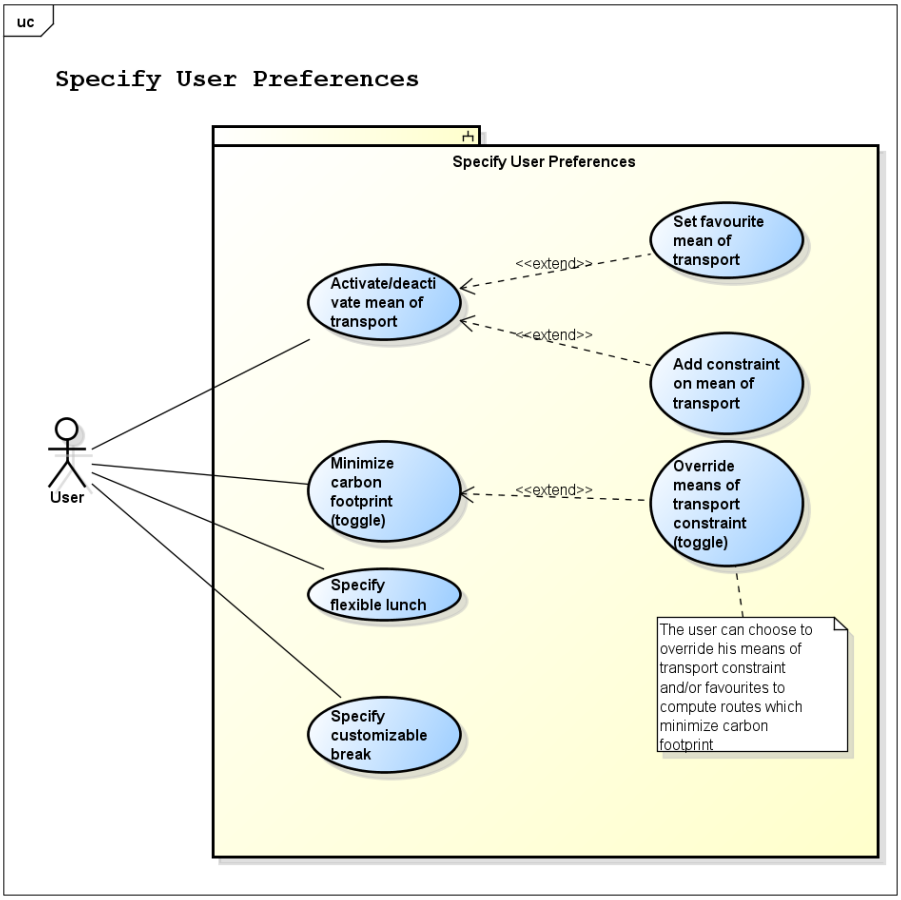

Figure 12: Use cases showing the user preferences and relative functonalities

3.4.10 Means activation/deactivation

Name	Activate/deactivate mean of transport
Actors	User
Entry conditions	The user is logged in and is in the user preferences page
Flow of events	• The user clicks on "Choose means of transport" button and he is redirected to a page containing a list of all possible means of transport; • The user unflags all the means of transport he does not intend to use. • The user clicks on "Done" button and is redirected to the user preferences page.
Exit conditions	The unflagged means of transport are removed from the possible means needed to compute a route
Exceptions	The user unselected every mean of transport, clicking "Done" button has no effect and an error message stating that at lest one mean of trasport has to be flagged.

Table 9: Activation/deactivation of a means Use Case table

3.5 Activity Diagrams

3.5.1 Meeting creation process

Whenever the user fills up the form relative to creation of a meeting and presses "Create meeting" the system does what is visually described in the diagram. It checks for conflicts while asynchronously queries public transport providers, when the data gets back and the meeting is set as regular or is put in a warning with other meetings the meeting is created and the process is done.

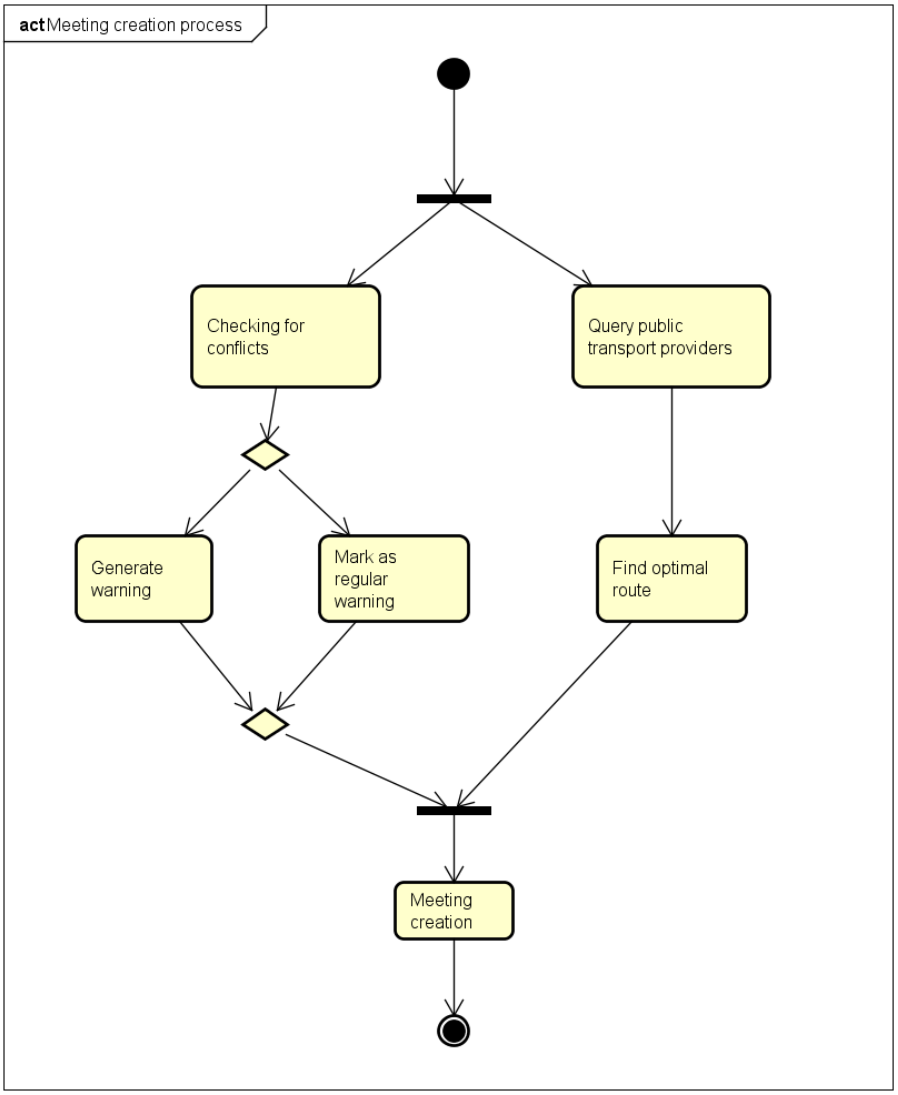

Figure 13: This diagram shows the activities carried out by the system in the meeting creation process

3.6 State Charts

3.6.1 Meeting State Machine

This State Machine was created with the purpose to identify the various states a meeting can be in and the show the transition events that modify its state.

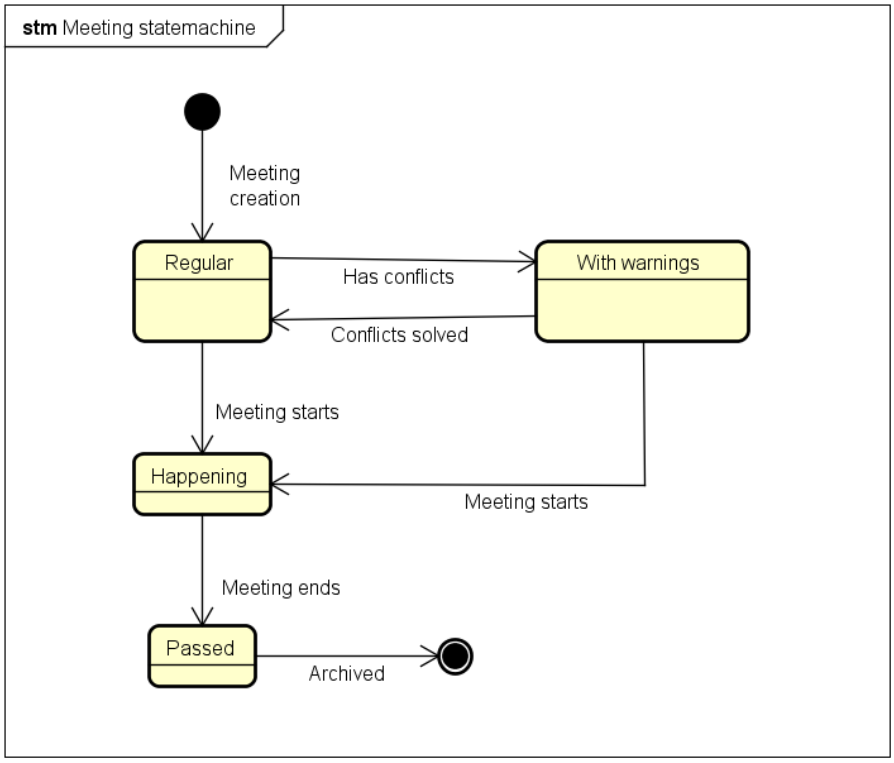

Figure 14: State Chart showing states of a meeting

3.6.2 Basic UX State Chart

We provide a simple and basic User Experience State Chart which highlights the different pages a user can find himself in (and the System redirects the user to, when an event happens).
We believe this could be helpful to understand and visualize the entire application.

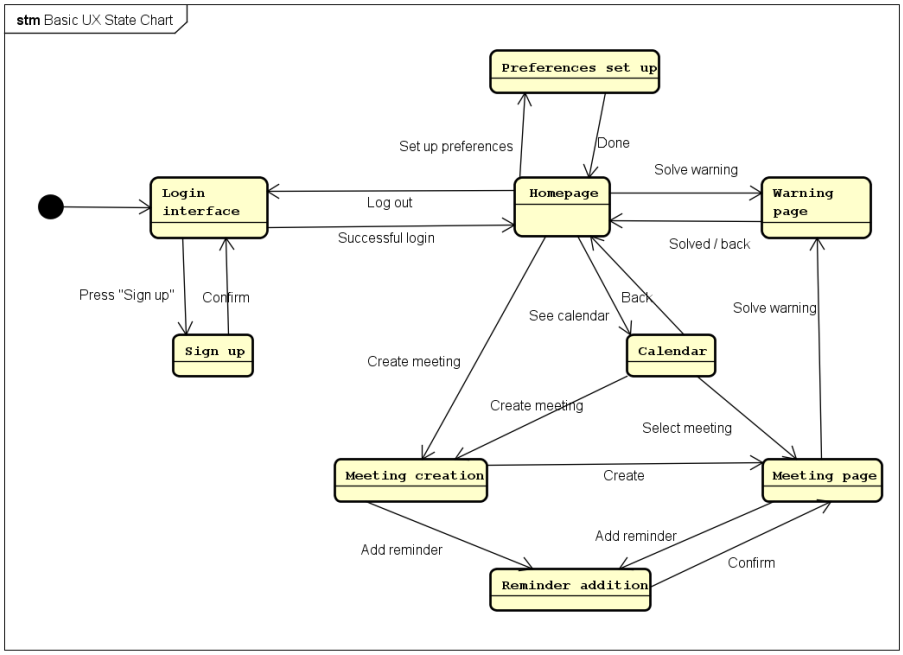

Figure 15: A basic User eXperience chart of the application

3.7 Sequence Diagrams

3.7.1 Signup/Login Sequence Diagram

This sequence diagram points out the steps to make a proper signup and to correctly login to Travlendar+ as a specific user. As you can see, there are three main phases, fullfilling the registration form, verifying the user email and logging in. In order to mark up the difference between registered and unregistered customers, we called them respectively 'user' and 'guest'. Clearly, the email verification leads to an automatic login, thus the system redirect the user to his personal page. This also explain why we have inserted logout before the login step.

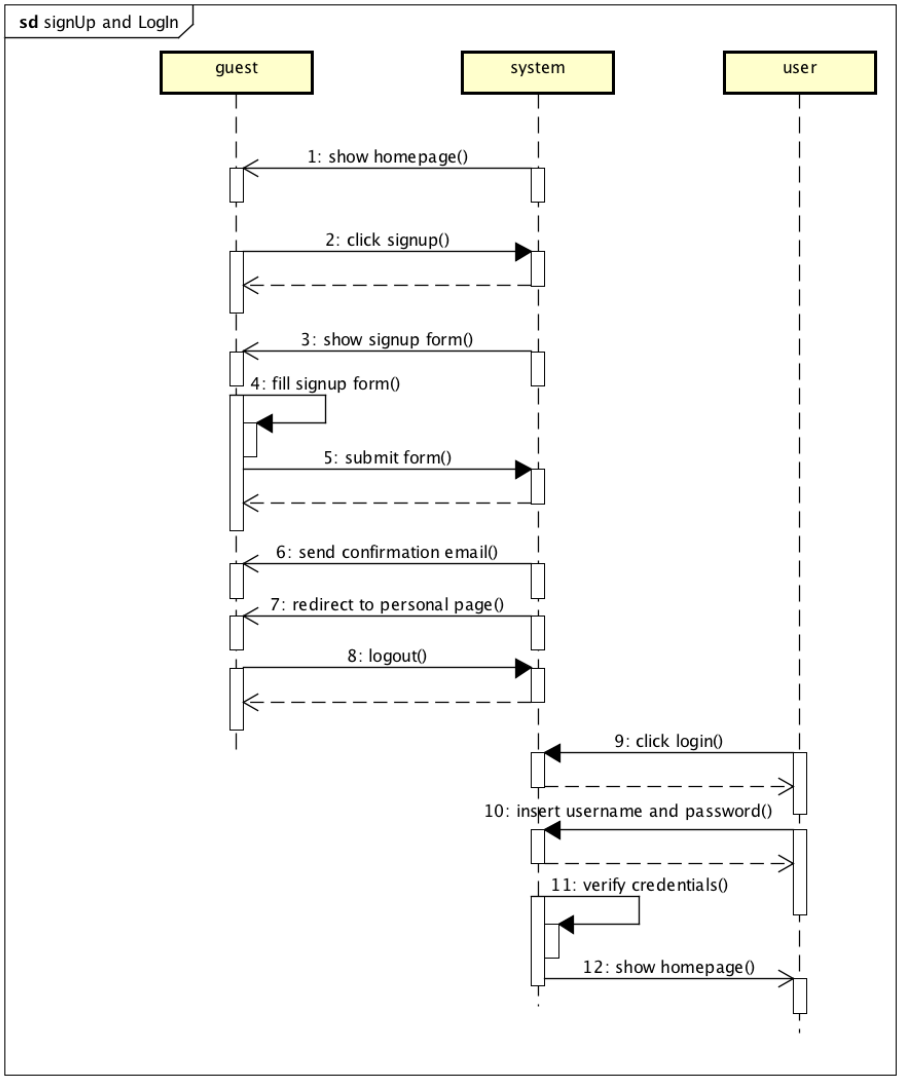

Figure 16: The sequence diagram about Signup and Login operations.

3.7.2 Meeting Creation Sequence Diagram

This sequence diagram explains how to create a meeting, assuming that the operation is accomplished properly. Oppositely,in case of failures such as trying to insert a duplicate meeting, Travlendar+ prevents the appointment registration. As the chart shows, when a user select a date on the calendar, the system provides him a meeting form, where he can insert all the required information. Then, the app. enters in an intense computational phase. In addition to the local operations such as verifying overlappings and analyzing preferences, during this step Travlendar+ also interacts with external systems. We designed them with the label 'service providers' referring for example to the public transports server, a forecast agency infrastructure, sharing services systems etc, and we suppose the related events to be database queries. The entire process ends with the registration of the event into the system and a positive notification to the user.

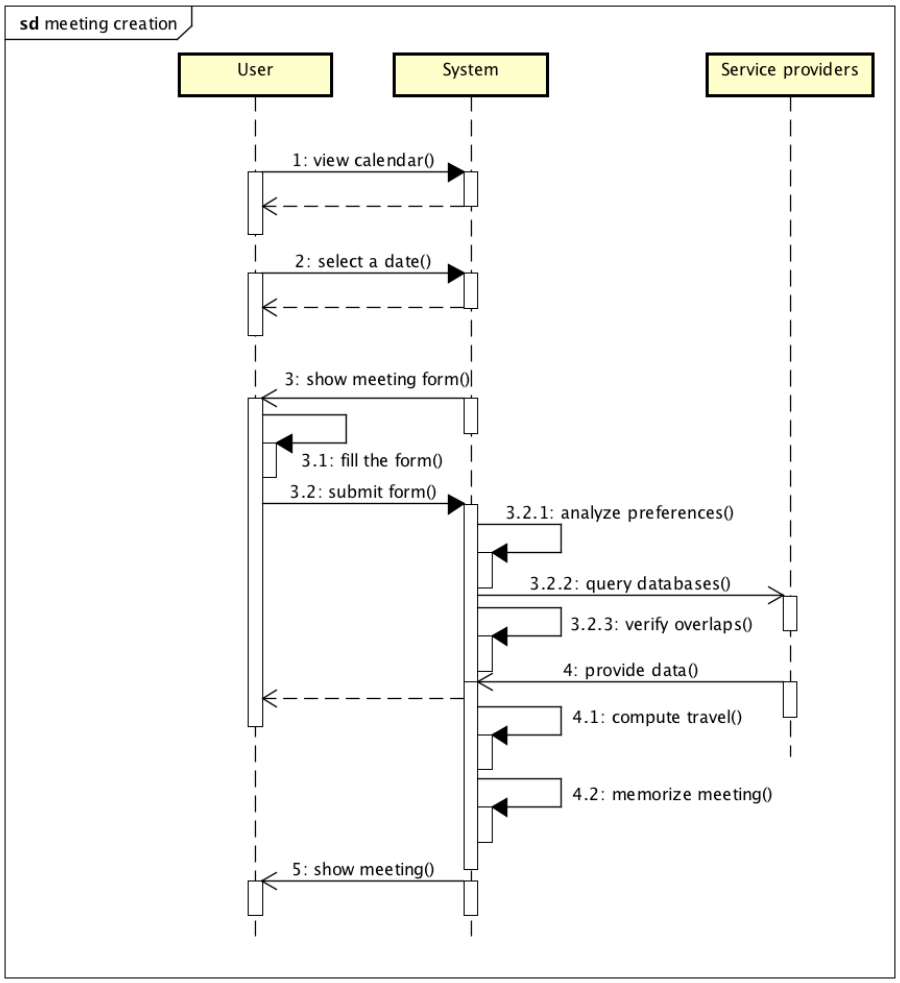

Figure 17: The sequence diagram about the process flow of a meeting creation.

3.7.3 Warnings Solving Sequence Diagram

The last two sequence diagrams refer to the user reaction against the generation of a warning by the system. In order to model correctly the two possibile user choices, we have done two diagrams, one for each opportunity. The first and simply chart explains what happens if the user decides to ignore a warning, that is just the deletion of the notification, because we assume his specific intention not to consider it. The second diagram, quite more complex, regards the warning resolution through an event rescheduling made by the user. For this reason, Travlendar+ allows to directly go from the warning to the conflictual meetings, in order to modify them. Obviously, considering that a conflict, same as a warning, can involve from two up to a huge number of appointments, it is clear that the operation number 5 could be done for more than just one meeting involved in the conflict.

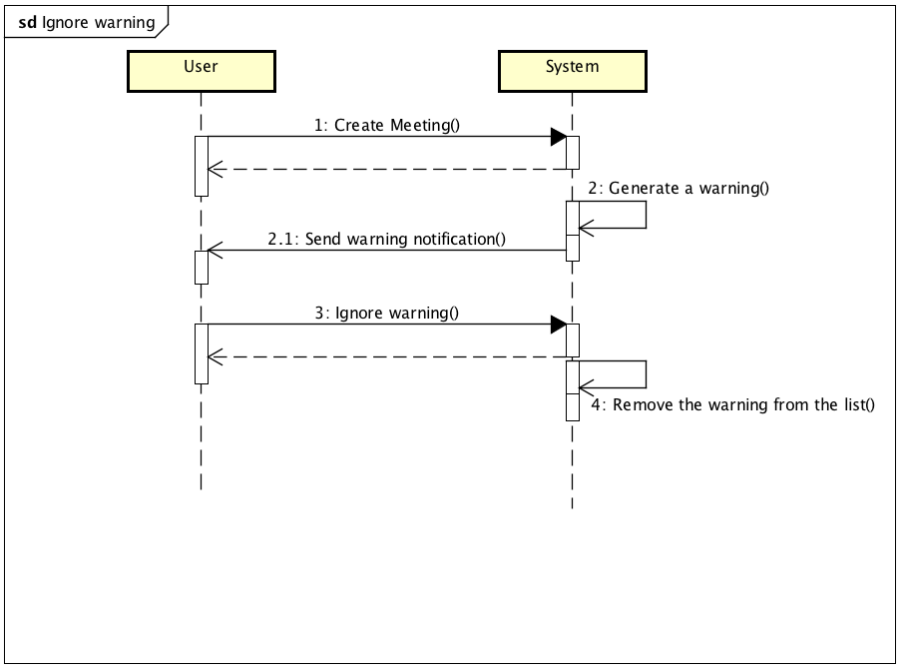

Figure 18: The sequence diagram that shows the steps to ignore a warning.

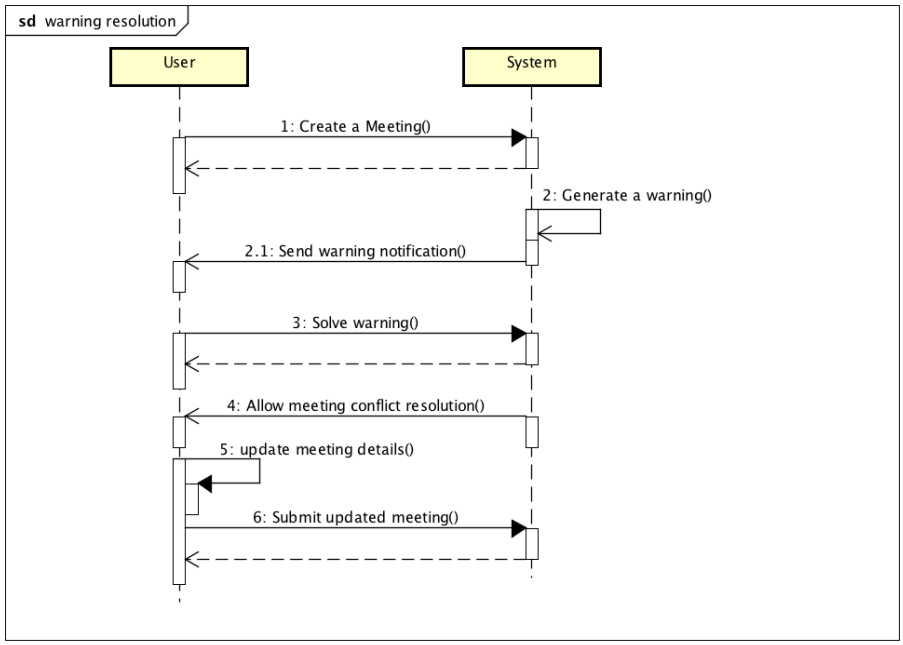

Figure 19: the sequence diagram that shows how to solve a warning.

3.8 Class Diagram

In order to convey detailed and clear information, we provide a high-level representation of Travlendar+ skeleton of the software side, however this is just a first sketch that will be enriched and improved or even changed in the design document.

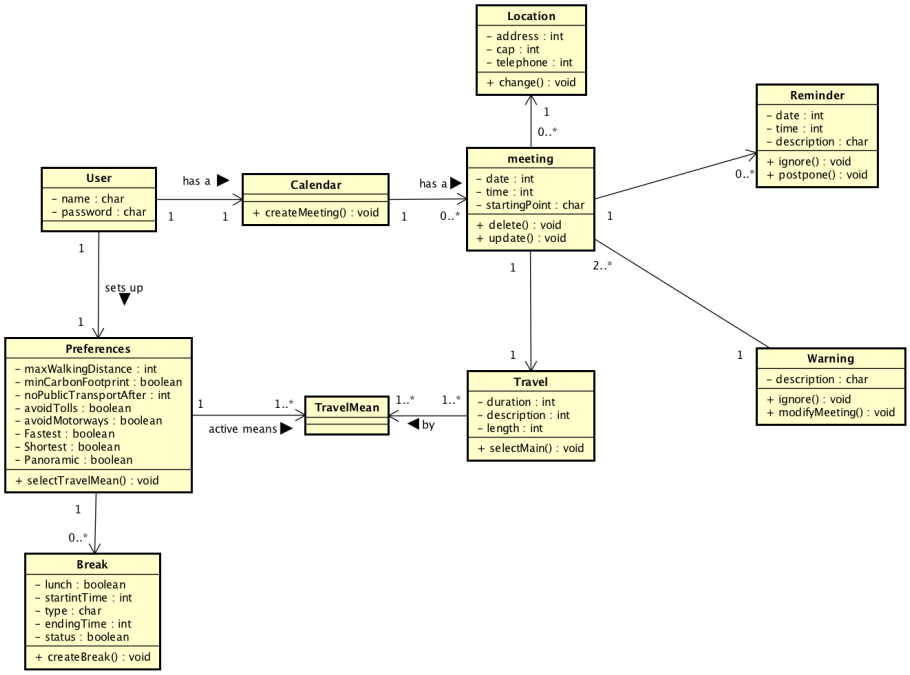

Figure 20: Travlendar+ class diagram

3.9 Traceability Matrix

RawID	**GoalID**	FunctionsID	ConstraintsID	UseCasesID	Comments
MM1	G1	F2	C4, C5, C6, C7	Table 6	Figure 13, Figure 17
MM2	G2	F6, F8	C10, C11	-	-
MM3	G3	F4	-	Table 8	-
MM4	G4	F3	C10, C11, C13	Table 9, Figure 12	-
MM5	G5	F7	C12	Table 7	-
MM6	G6	F9	C4, C5, C6, C7	-	Figure 11, Figure 14
MM7	G7	F5	C12	Figure 11	-
MM8	-	F1	C1, C2, C3	Figure 10, Table 3, Table 4	-

Table 10: Traceability table

4 Formal Analysis using Alloy

4.1 Model

```
1   //****************SIGNATURES*****************//
2
3   abstract sig Boolean {
4
5   }
6
7   one sig true extends Boolean { }
8
9   one sig false extends Boolean { }
10
11
12  sig Travel {
13      means:  set TravelMean
14  }
15
16  abstract sig TravelMean { }
17
18  one sig Car extends TravelMean {
19
20  }
21
22  one sig SharedCar extends TravelMean { }
23
24  one sig Bike extends TravelMean { }
25
26  one sig SharedBike extends TravelMean { }
27
28  one sig Train extends TravelMean { }
29
30  one sig Metro extends TravelMean { }
31
32  one sig Bus extends TravelMean { }
33
34  one sig OnFoot extends TravelMean { }
35
36  sig Preferences {
37      minimizeCarbonFootprint:  one Boolean,
38      breaks:  set Break,
39      activeMeansOfTransport:  TravelMean -> one Boolean
40  }{
41      // A set of preferences must refers only to one user
42      one u:User | this = u.preferences
```

```
43      }
44
45      sig Break {
46          active:  one Boolean,
47          lunch:  one Boolean
48      }
49
50      sig User {
51          calendar:  one Calendar,
52          preferences:  one Preferences
53      }
54
55      sig Calendar {
56          meetings:  set Meeting
57      }{
58          // A calendar must refers only to one user
59          one u:User | this = u.calendar
60      }
61
62      sig Reminder {
63
64      }
65
66      sig Location {
67
68      }
69
70      sig Meeting {
71          location:  one Location,
72          route:  one Travel,
73          next:  lone Meeting,
74          previous:  lone Meeting,
75          reminders:  set Reminder,
76          status:  one MeetingState,
77          warning:  lone Warning,
78          conflict:  set Meeting,
79          weather:  one WeatherForecast
80      }
81
82      abstract sig MeetingState {
83
84      }
85
86      one sig Regular extends MeetingState{ }
87
88      one sig WithWarning extends MeetingState{ }
```

```
 89
 90      one sig Happening extends MeetingState{ }
 91
 92      one sig Passed extends MeetingState{ }
 93
 94      abstract sig WeatherForecast {
 95
 96      }
 97
 98      //This symbolizes every case of 'good' weather (e.g.  Cloudy,
Partially Cloudy, etc.)
 99       one sig Sunny extends WeatherForecast{ }
100
101      //This symbolizes every case of 'bad' weather (e.g.  Snowy,
Stormy, extremely Windy, etc.)
102       one sig Rainy extends WeatherForecast { }
103
104      //doppia freccia tra meeting e warning
105
106      sig Warning {
107          conflicts:  set Meeting
108      } {
109      //meetings set has to be at least >=2     otherwise the conflict
wouldn't exist
110          #conflicts > 1
111      }
112
113      //problema:  tutti i meeting nella stessa location
114
115      //****************CONSTRAINTS*****************//
116
117      //-----------------MEETING STATUS CONSTRAINTS
118
119      //The status=With Warning implies that there is a Warning
120      fact {
121          all m:Meeting | m.status=WithWarning implies #m.warning=1
122      }
123
124      //Every Meeting with a warning must have their status set
to With Warning
125      fact {
126          all w:Warning | all m:w.conflicts | m.status=WithWarning
127      }
128
129      //If the status is Regular, Happening or passed, the Meeting
does not have any Warning
```

```
130     fact {
131         all m:Meeting | (m.status=Regular or m.status=Passed or
m.status=Happening )
132                     implies #m.warning=0
133     }
134
135     //-----------------------CARDINALITY CONSTRAINTS
136
137     //Function to retrieve the Calendar owner
138     fun calendarOwner[c:  Calendar]:  set User {
139         {u:  User | u.calendar = c}
140     }
141
142     //A User can only have one Calendar
143     fact OneCalendarPerUser{
144         no disj c1,c2:  Calendar | all u:User | c1     in u.calendar
and c2     in u.calendar
145     }
146
147     //Different Users can't have the same Calendar
148     fact noSameCalendarDifferentUsers{
149         no disj u1,u2:  User | some c:  Calendar | c in u1.calendar
and c in u2.calendar
150     }
151
152     //It can't exist a Calendar not associated to any User
153     fact noCalendarWithoutUser{
154         all c:Calendar | some u:User| u.calendar=c
155     }
156
157     //It can't exist a Break not associated to any Preferences
158     fact noBreakOutsidePreferences{
159         all b:Break | some p:Preferences| b in p.breaks
160     }
161
162     //It can't exist a Location not associated to any Meeting
163     fact noUnneededLocation{
164         all l:Location | some m:Meeting | m.location=l
165     }
166
167     //One meeting can be in only one calendar
168     fact MeetingInOnlyOneCalendar {
169         no disj c1,c2:Calendar | some m:Meeting | m in c1.meetings
and m in c2.meetings
170
171     }
```

```
172
173     //For every Meeting there exists one and only one Calendar
containing it
174     fact noMeetingOutsideCalendar{
175         all m:Meeting | some c:Calendar | m in c.meetings
176     }
177
178     //A reminder is unique for a Meeting
179     fact uniqueReminderForMeeting {
180         no disj m1,m2:  Meeting | some r:Reminder | r in m1.reminders
and r in m2.reminders
181     }
182
183     //Every reminder is associated to a meeting
184     fact reminderIsSetForAMeeting{
185         all r:Reminder | some m:Meeting | r in m.reminders
186     }
187
188     //There exists at least one Travel Mean for any Travel
189     fact atLeastOneTravelMean{
190         all t:Travel | some m:  TravelMean | m in t.means
191     }
192
193     //-----------------WEATHER CONSTRAINTS
194
195
196     //If the forecasts say it's going to rain (WeatherForecast=Rainy)
in the date of a Meeting, the Travel
197     //associated to that Meeting can't contain the following means
of transport:  OnFoot, Bike, SharedBike
198     fact dontWetUsers {
199         all m:Meeting | m.weather=Rainy implies (OnFoot not in
m.route.means and
200                     Bike not in m.route.means and SharedBike not
in m.route.means)
201     }
202
203     //----------------------MEETINGS CONSTRAINTS
204
205     fact precedenceRelationConstraint {
206         next= previous
207     }
208
209     //For every Meeting in any Calendar, the previous (and therefore
also the next) is in that Calendar
210     fact{
```

```
211         all m:Meeting | all c:Calendar | m in c.meetings implies
m.previous in c.meetings
212     }
213
214     //A Meeting can't have itself as next or previous meeting
nor it can be next
215     //of the next and so on (transitive closure), (previous is
automatically
216     //verified because of the precedenceRelationConstraint fact
217     fact precedence{
218         no m:  Meeting | m in m.^next
219     }
220
221     //If a Meeting is in conflict with another one, the second
one must be in conflict with the first one
222     fact conflictualMeeting{
223     all disj m1, m2:  Meeting | m2     in m1.conflict implies
m1     in m2.conflict
224     }
225
226     //A Meeting can't be in conflict with itself
227     fact noAutoConflict{
228         no m:  Meeting | m in m.conflict
229     }
230
231     //There can't be a conflict between two Meetings if they are
in the same location
232     fact noConflictBetweenSameLocationMeetings{
233         all disj m1,m2:  Meeting | all l:Location | (m1.location
= l and m2.location= l ) implies
234         (m2     not in m1.conflict and m1     not in m2.conflict)
235     }
236
237
238     //---------------------WARNING CONSTRAINTS
239
240
241     //Every Meeting in his conflicts set must have the next Meeting
or the Previous Meeting in the set
242     fact adjacentConflicts{
243         all m:  Meeting | all m2:  m.conflict | m2.previous in
m2     or m2.next in m2
244                                       or m2     = m.previous or
m2     = m.next
245     }
246
```

```
247     //It can't exist a Meeting in a Warning if this Meeting isn't
in conflict with some other Meeting
248     fact noWarningWithoutConflicts{
249         all disj m1,m2:Meeting |some w:Warning | m1     in w.conflicts
and m2     in w.conflicts implies m1     in m2.conflict and m2     in
m1.conflict
250     }
251
252     //If there exists 2     Meetings in conflict with each other,
there exists a Warning containing both of them
253     fact warningExistence{
254         all disj m1,m2:  Meeting | m1     in m2.conflict implies
one m1.warning and
255                             one m2.warning and (some w:  Warning
| m1     in w.conflicts
256                                                 and m2     in
w.conflicts)
257     }
258
259     //For every pair of disjoint Meetings contained in a Warning,
one is in the conflict set of the other and viceversa
260     fact {
261         all w:  Warning | all disj m1,m2:  w.conflicts | m1     in
m2.conflict and m2     in m1.conflict
262     }
263
264     //A meeting can only be contained in one Warning
265     fact exclusiveWarning{
266         all m:Meeting | all disj w1,w2:Warning | m in w1.conflicts
implies m not in w2.conflicts
267     }
268
269     //There can only exist a conflict between Meetings in the
same Calendar
270     fact onlyConflictsInSameCalendar{
271         all disj m1,m2:Meeting | some c:Calendar | m1     in m2.conflict
and m2     in m1.conflict implies m1     in c.meetings and m2     in
c.meetings
272     }
273
274     //Empty conflict set implies empty warning set and viceversa
275     fact {
276         all m:Meeting | #m.conflict=0     implies #m.warning=0
277     }
278
279     //All meetings in a warning conflicts have to be in conflict
```

```
with each other
 280     fact noWarningIfNoConflicts{
 281         all w:Warning | all disj m1,m2:  w.conflicts | m1     in
m2.conflict and m2     in m1.conflict
 282     }
 283
 284     //If a meeting is in a warning, the warning has the meeting
in its conflicts
 285     fact meetingWarningRelationCorrespondance{
 286         all w:Warning | all m:Meeting | (m in w.conflicts) implies
(w in m.warning)
 287     }
 288
 289
 290
 291     //---------------------PREFERENCES CONSTRAINTS
 292
 293
 294     //In every preferences at least one travel mean has to be
selected
 295     fact atLeastOneSelectedTravelMean{
 296         all p:Preferences | some t:TravelMean |one tr:true | tr
in t.(p.activeMeansOfTransport)
 297     }
 298
 299     //A break has to be in one and only one Preferences
 300     fact noEqualBreaks{
 301         no disj p1,p2:Preferences | some b:Break | b in p1.breaks
and b in p2.breaks
 302
 303     }
 304
 305     //---------------------TRAVELS CONSTRAINTS
 306
 307     //Meetings happening in different locations can't have the
same travel to get there
 308     fact noDifferentLocationsSameTravelMeetings{
 309         all disj m1,m2:  Meeting | (m1.location != m2.location)
implies (m1.route != m2.route)
 310
 311     }
 312
 313
 314     //There cannot exist a travel not associated to any meeting
 315     fact noDisassociatedTravel {
 316         all t:Travel | some m:Meeting | m.route=t
```

```
317     }
318
319     //The travel must include only travelMeans that are selected
in the preferences
320     fact onlyUseActiveTravelMeans{
321         no t:TravelMean | some u:User | some p:u.preferences |
some c:  u.calendar |
322     some m:c.meetings | some r:  m.route |
323     some f:false | t in r.means and f in t.(p.activeMeansOfTransport)
324
325     }
326
327
328
329     //If a travel mean is deactivated, it can't be in any travel
in any meeting
330
331
332     //*****************ASSERTIONS*****************//
333
334     assert singleUserCalendar {
335         all c:Calendar | one u:User | u.calendar = c
336     }
337
338     check singleUserCalendar for 2
339
340     //There can't exist 2     warning with the same meeting in
the conflicts set
341     assert noMoreWarningSameMeeting {
342         no disj w1,w2:  Warning | some m:Meeting | m in w1.conflicts
and m in w2.conflicts
343     }
344
345     check noMoreWarningSameMeeting for 3
346
347     assert usersAreNeverGettingWet {
348         no m:  Meeting | m.weather=Rainy and ((Bike in m.route.means
or SharedBike
349                                 in m.route.means or OnFoot in
m.route.means))
350     }
351
352     check usersAreNeverGettingWet for 4
353
354     assert neverUseInactiveTravelMeans{
355         no u:User | some t:TravelMean | some travel:  u.calendar.meetings.route
```

```
|
 356         false in t.(u.preferences.activeMeansOfTransport) and
t in travel.means
 357
 358     }
 359
 360     check neverUseInactiveTravelMeans
 361
 362
 363     assert noConflictualMeetingsOutsideWarning{
 364         all disj m1,m2:Meeting | some w:Warning | m1     in m2.conflict
implies m1     in w.conflicts and
 365                                       m2     in w.conflicts
 366     }
 367
 368     check noConflictualMeetingsOutsideWarning
 369
 370     //****************PREDICATES****************//
 371
 372     pred showLotsOfMeetings{
 373         #User=1
 374         #Meeting=8
 375     }
 376
 377     run showLotsOfMeetings { } for 3     but exactly 8     Meeting,
exactly 1     User, exactly 2     Warning
 378
 379     pred showMoreUsers{
 380         #User=2
 381         #Meeting=4
 382     }
 383
 384     run showMoreUsers { } for 2     but exactly 4     Meeting,
exactly 2     User
 385
 386     pred showNoMeetingsInstance {
 387         #Meeting=0
 388     }
 389     run showNoMeetingsInstance { } for 6     but exactly 6     User,
exactly 0     Meeting
```

4.2 Results

Here we provide a screenshot of the results derived from the execution of all the runs over possible predicates and checks on assertions using Alloy

Analyzer.

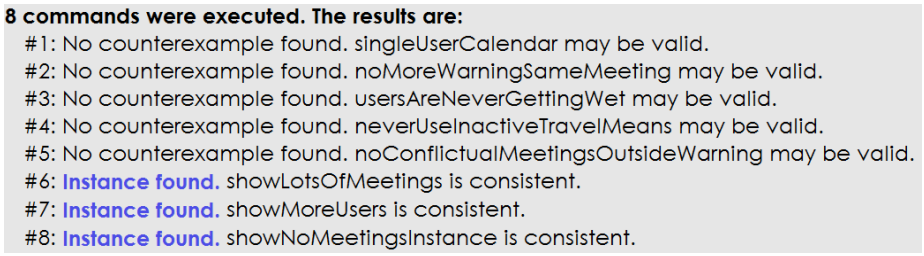

Figure 21: The results of the alloy model

4.3 Worlds Generated

4.3.1 World 1

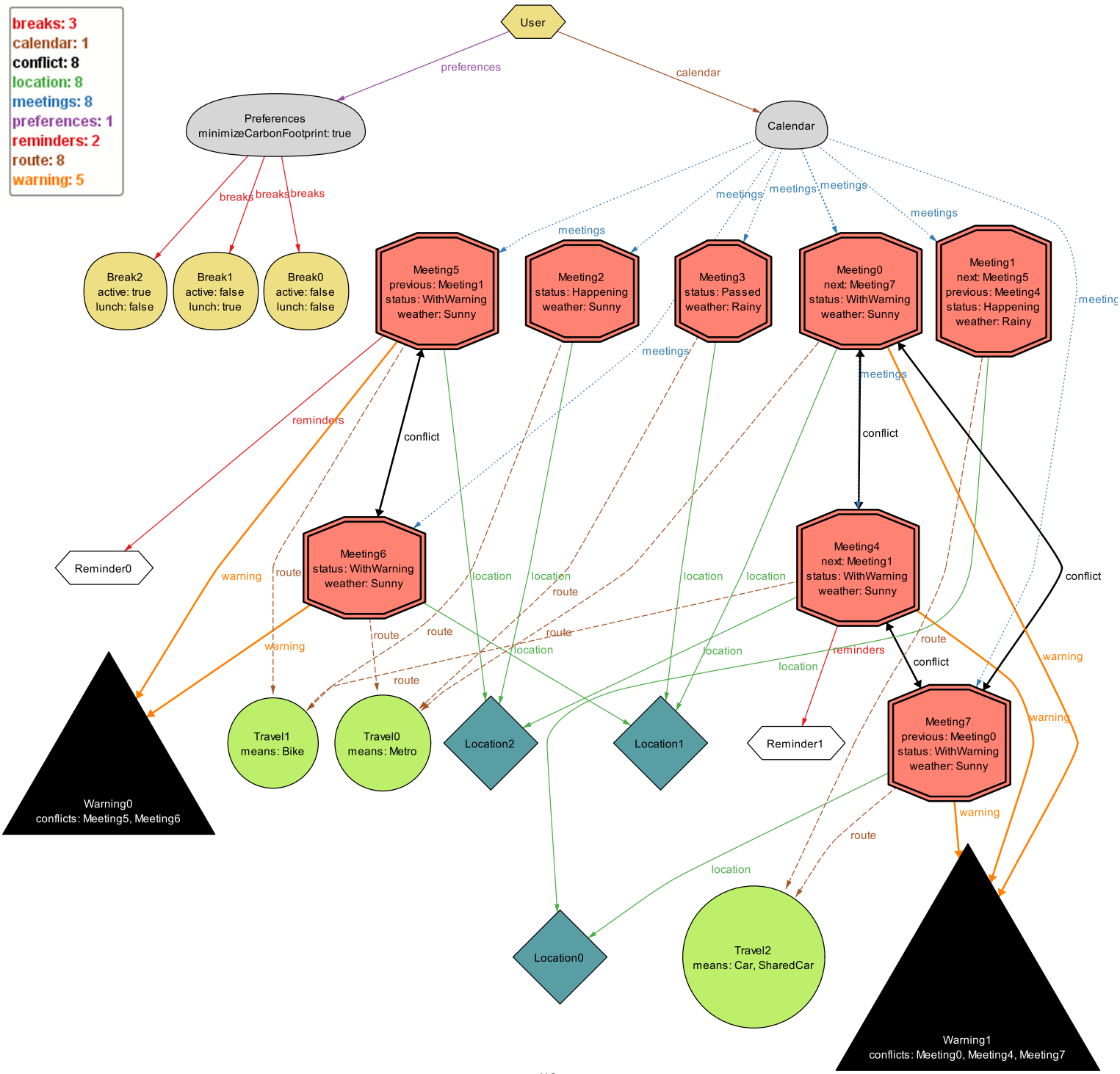

Figure 22: Alloy World 1

4.3.2 World 2

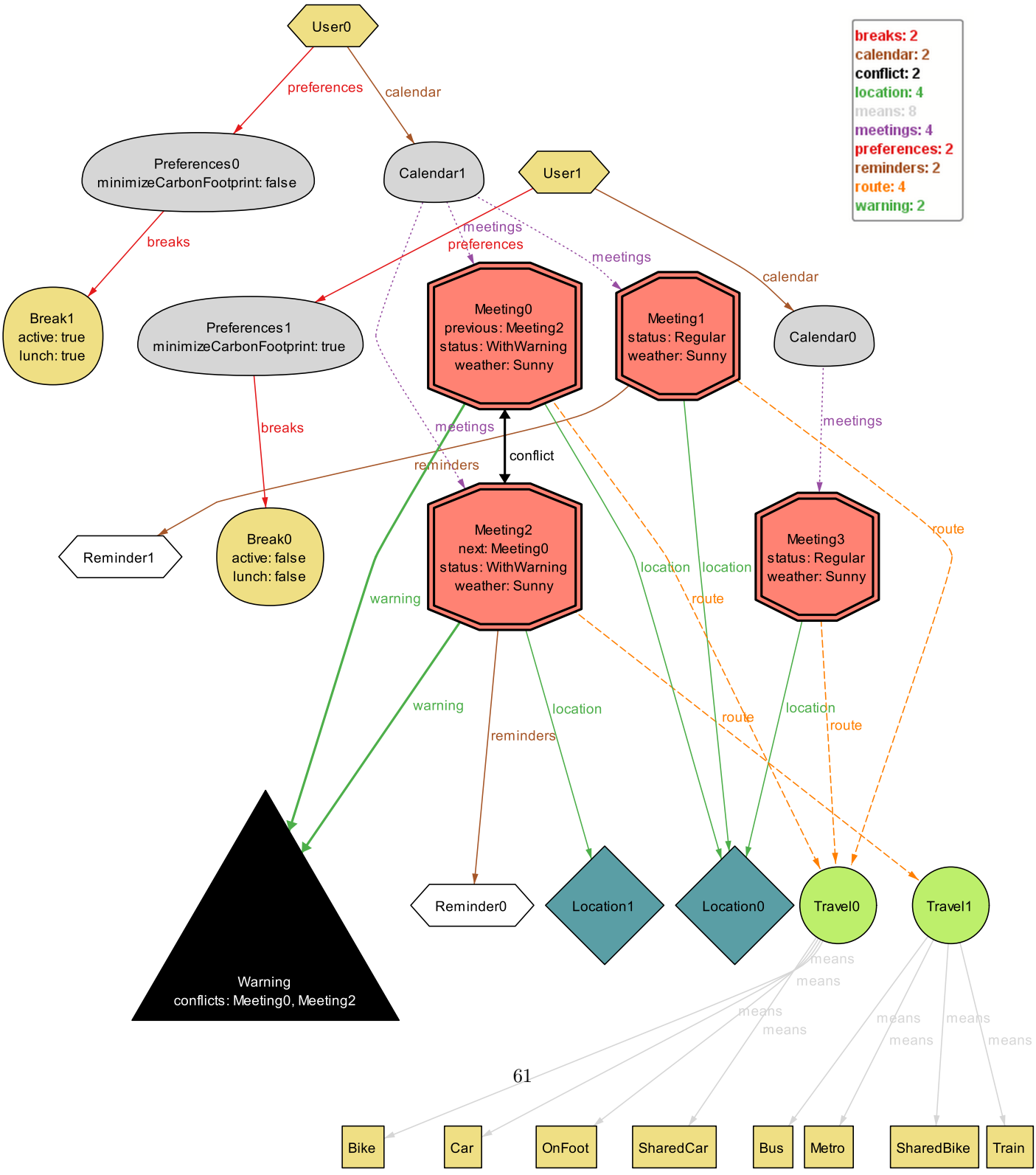

Figure 23: Alloy World 2

5 Appendix

5.1 Used software

Task	Software
Edit and compile LaTeX code	TeXmaker
UML modelling	Astah Pro, Signavio
Compile and run Alloy	Alloy Analyzer 4.0
Mockup creation	Balsamiq Mockups 3

5.2 Effort spent