Politecnico di Milano

Corso di Laurea Magistrale in Computer Science and Engineering
Dipartimento di Elettronica e Informazione

Travlendar+

Requirement Analysis and Specification Document

Reference professor:
Prof. Elisabetta Di Nitto

Authors:
Alessandro Aimi Matr. 899642
Roberto Bigazzi Matr. 899411
Filippo Collini Matr. 829918

Version 1.2.0
07/01/2018
Academic Year 2017-2018

Contents

Chapter 1

Introduction

This document is the Requirement Analysis and Specification Document (RASD) of a mobile application called Travlendar+. The purpose of the document is to show the requirements and specification of the new application, considering various aspects like the stakeholders' needs, domain properties and constrains which the system-to-be is subject to.

1.A Description of the given problem

Travlendar+ is a mobile, calendar-based application that helps the user to manage his appointments and to a greater extent set up the trip to his destination, choosing the best means of transport depending on his needs.
Travlendar+ will choose the most suitable way to get the user to his destination between a large pool of options, considering public transportation, personal vehicles, locating cars or bikes of sharing services and walking to the destination. It will take account of weather, traffic, possible passengers if any, the user-set break times and the potential will to minimize the carbon footprint of the trip, always focusing on taking him on time to his scheduled appointments.
Eventually the user will be able to purchase the tickets he will use to reach his destination in-app. The great customizability is one of the main strengths of Travlendar+, being able to fully comply with the user needs.

1.B Definitions and Acronyms

1.B.1 Definitions

- Application: it depends on context, but often it is referred to Travlendar+;
- User: see 2.A - Actors;
- Guest: see 2.A - Actors;
- System: see Application;
- Event: often used as a key word referring to the events the user can add on the calendar embedded in Travlendar+;
- Trip: often used as a key word referring to the trips alternatives the system processes and suggests the user;
- Dynamic event: Event whose location in time can be decided by the application on various factors;
- Means of transport: often used as a key word referring to the means of transportation the user can choose in the settings of Travlendar+;
- Private transports: often used to mean trains, planes and all private companies transports;
- Shared means: used to mean cars and bicycles of car and bike sharing companies.

1.B.2 Acronyms

List of the acronyms used in this paper:

- RASD: Requirements analysis and specification document;
- ETA: Estimated time of arrival, it's the time remaining to arrive to destination;
- POI: Point of interest;
- API: Application programming interface;
- UI: User Interface.

1.C Revision History

- 29/10/2017 Version 1.0.0 - First complete drawing up of the RASD;
- 18/11/2017 Version 1.0.1 - Fixed some grammatical errors;
- 24/11/2017 Version 1.1.0 - Changed Sections 3.B.4 - Communication interfaces and 3.F.3 - Security;
- 26/11/2017 Version 1.1.1 - Added some acronyms and modified requirement 34;
- 07/01/2018 Version 1.2.0 - Some alloy improvements.

1.D References

Documents list:

- Mandatory Project Assignments.pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- RASD sample from Oct. 20 lecture.pdf

Online sites list:

- http://alloy.mit.edu/alloy/documentation.html

1.E Document Structure

The paper is structured as follows:

- Chapter 1: Introduction to the document, including some information about its composition and the description of the problem;
- Chapter 2: General description of the system functions and assumptions, together with information about the environment and the users;
- Chapter 3: Detailed description of the functional, non functional requirements and constraints;
- Chapter 4: System usage examples;

- Chapter 5: Specific modeling of system functions, abstract structure and usage;
- Chapter 6: Formal analysis of the system and his functions using Alloy;
- Chapter 7: Effort spent by the authors to draw up the document.

1.F Used tools

The tools used to create this document are:

- StarUML for UML mode;
- Alloy Analyzer 4.2 for proving consistency of the model and to challenge it to find counterexamples for our assertions;
- Github as version controller and to share documents;
- LaTeX for typesetting this document;
- Texmaker as editor
- Draw.io to create sequence diagrams and state diagrams;
- Signavio to create use case diagrams;
- Photoshop for mockups.

Chapter 2

Overall Description

2.A Actors

- Guest: Person that's not registered yet. He can see the login and the registration page.
- User: Person registered and logged to the app. He can access to his calendar to see his appointments, setup new meetings or manage them, change his preferences, ...

2.B Goals

The aim of the system is to give the user the following functionalities:

- registration to the service and preferences set up;
- anytime management of personal and mobility preferences;
- create and schedule a new event choosing time and location;
- edit previously added event data;
- change travel options on automatically created trips to/from event;
- mobility companies' tickets purchase via built-in browser;
- provide high customizability of personal preferences regarding user daily routines.

and to provide on its own these other ones:

- manage in a clever way trips to/from user events, relying on the map of the surroundings, time of the day, public and private transports' timetables and stops, shared means, traffic, weather forecast, possible strikes, event information and user preferences;
- notify the user on time of upcoming trips;
- warn him in case of an exception.

2.C Domain Assumptions

We suppose that the following conditions are true in the analysed world:

- the geographical area of the city is included in the coverage area of most common mobile communication technologies (3g, 4g) offered by main telecommunications companies;
- users must be subscripted to a sharing service if they want to use it;
- APIs used by the application will always be updated on traffic status, eventual incidents and weather conditions;
- sharing services' APIs signals their means if and only if the means are where the APIs say, and they are not occupied or booked;
- users always have a working internet connection;
- half an hour is enough warning time for users to start a trip;
- the user cannot ride two means of transport at the same time.

2.D Product Perspective

Travlendar+ will be developed as a mobile application that relies on the use of Google maps and Google calendar APIs.
Its user interface will be composed by two main tabs, one with a calendar, to schedule user's events and the other one with a map to manage the movements of the user.
In the future the system will employ APIs to buy tickets without using the built-in browser and it will have a service of technical assistance via chat.
The application will not provide any API for integration with other systems.

2.E User Characteristics

The user of the system-to-be can be every person who wants to schedule appointments in a calendar and manage his movements from a location to another at the same time. Users can use it to organize work events, but also include family or spare time events. The application doesn't have any age limit, or any other restriction applied to the user characteristic. In order to make the application work without limitation the user need to have access to the Internet, but he can access and modify the calendar offline.

Chapter 3

Specific Requirements

3.A Functional requirements

The aim of the system is to give the user the following functionalities, arranged in sections:

- Registration process:

    1. the application will provide a registration process;
    2. users can access to the functionalities provided by the system if and only if they register to it;
    3. there are not users with privileges, every registered user can access to the same features of the other ones and the system is safe even without supervisors;
    4. all data will be stored on the Travlendar+ server to backup user information and to allow it to be shared on different user's devices.

- Initial settings, always adjustable:

    - set user personal information:

        5. google account to synchronize calendar and maps;
        6. house location, work location, new location;
        7. default location to reach after appointments, to be chosen between the favourites (usually home);
        8. user break times and guaranteed amount of time to keep free from trips in every break to have lunch or another dynamic event;

9. time of the day after which bike (owned or shared) and public transports will not be considered anymore in planning the trip.

– set desired user transport means:

10. car possession, bike possession;
11. car sharing account(s), bike sharing account(s);
12. public and private transports (possibility to insert season ticket);
13. maximum walking distance (to destination or sharing vehicle).

- Scheduled event creation:

14. set day(s);
15. set time of beginning and end;
16. set location.

- Calendar consultation and edit:

17. see future scheduled events and meetings;
18. modify events previously added.

- Planned trips consultation and edit:

19. desire to minimize carbon footprint;
20. choose between any other trip possibilities in case the one recommended by Travlendar+ is not suitable;
21. purchase tickets for public transports before the trip;
22. presence of passengers;
23. show the best travel option selected by the app on various factors;
24. consult the map showing the best route to reach the destination, the sharing vehicle or the public transport stop.

- On arrival of scheduled meeting:

25. notification service informing the user he needs to leave to the next meeting in half an hour;
26. notification service telling the user the time to leave to the next meeting has come;

27. alert the user of trips' changes between the first reminder and the departure time.

- In case of exceptions:

28. warn the user of overlapping events creation and ask to choose primary event;
29. warn the user of possible inability to arrange trip that takes him in time to an unrechable future event.

- On trip planning:

30. the Google Maps API is used to determine best route (also considering traffic);
31. time of the day is considered on means of transport' choice;
32. public and private transports' timetables, stops and position (obtained through their APIs) are employed to evaluate their possible usage;
33. shared means position (obtained through their APIs) are employed to evaluate their possible usage;
34. weather forecasts are used to evaluate the possibility to go by bike or by foot;
35. possible strikes are considered on means' choice;
36. event information is used to set trip's destination, time of arrival or departure and location;
37. user preferences are considered to choose means and dynamic events allocation;
38. season pass information will be used to check the necessity of a ticket for the trip;
39. the best trip is processed for every means of transport not excluded;
40. the suggested trip is chosen according to the user trip preference (fastest, eco-friendly, customized).

3.B External Interface Requirements

3.B.1 User interfaces

The application will be developed as a mobile application for the main mobile operating systems (iOS and Android). Its user interface will appear

the same to all users, and must be user-friendly and intuitive. To show how the application will look like some mockups of the user interface are present in the document (mockups are realized for iOS operating system).

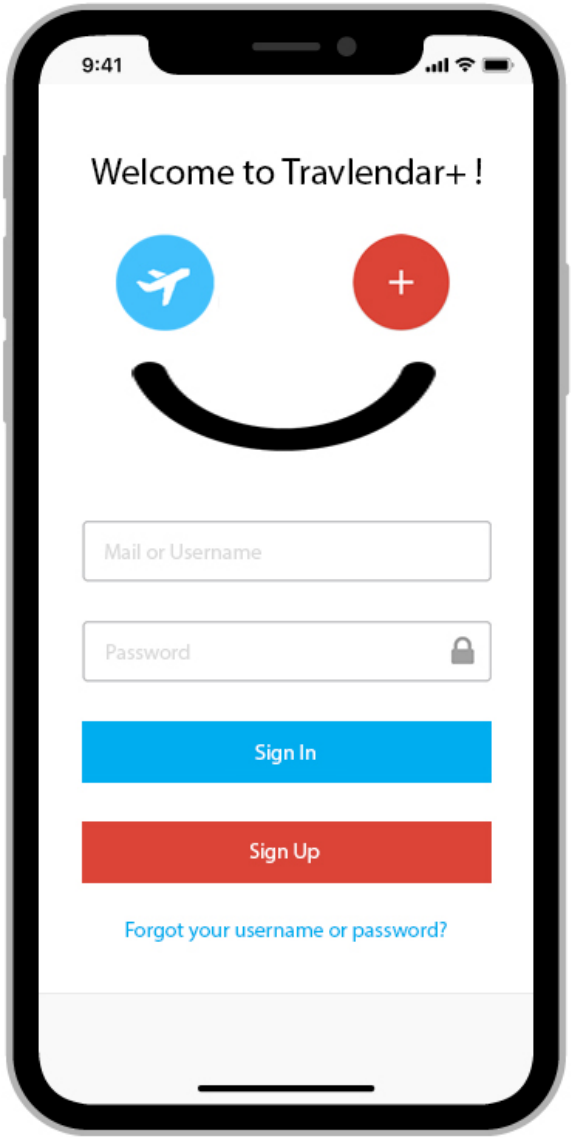

Figure 3.1: Mockup of the login screen

After downloading the application from the store of the OS. At the first start if the user is not registered, so it is a guest yet, he must sign up. Otherwise he can log in. If he is signing up he will see the registration screen.

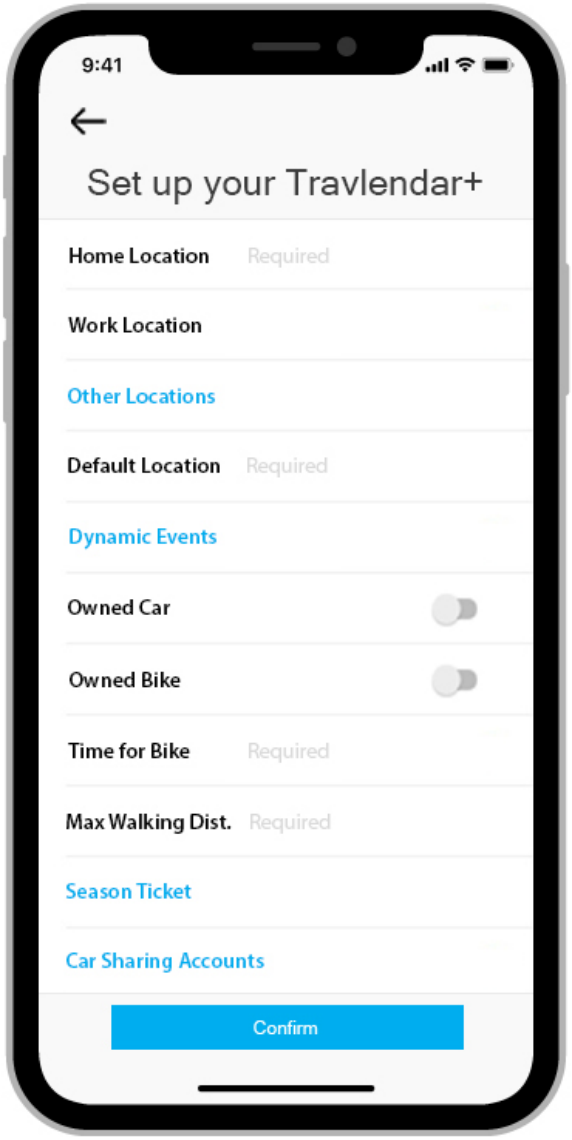

Figure 3.2: Mockup of the first configuration screen

The first time the user signs in, the application shows the first setup screen where the user has to insert all his preferences. The light blue texts open other screen where more are given for the specific setting.
Once the application is configured, the first screen that will appear to the user at each startup is the main screen that shows up a calendar, where he can add his events.

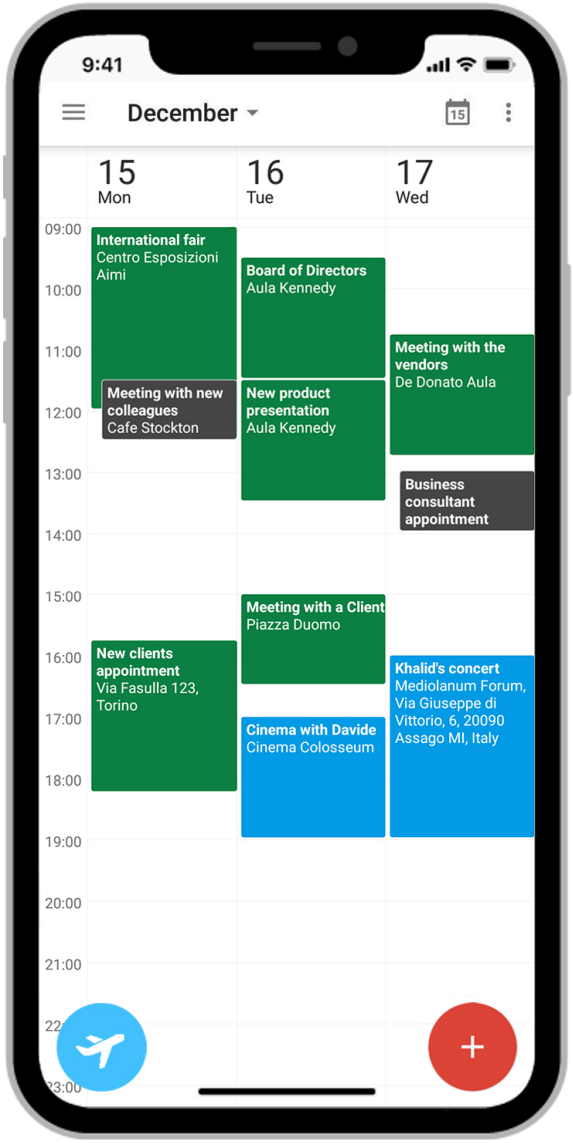

Figure 3.3: Mockup of the main screen (calendar)

The events are the ones in green or blue, some event are not coloured cause are overlapped with others or the user cannot phisically arrive on time for them (the user chooses what event is primary and the application considers it to organize the trips. When the user chooses to add an event pressing the "Add Event" button on the bottom right corner (the red one with a cross on it), the application shows a screen where he can insert the details of it.

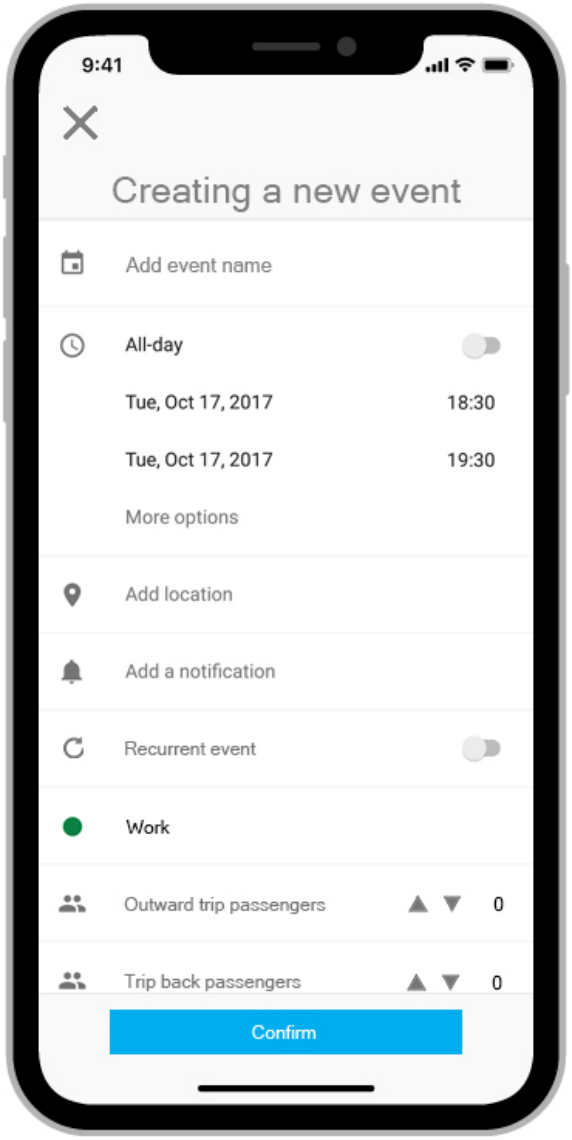

Figure 3.4: Mockup of the screen that appears to the user when he has to add an event

When the main screen is showed by the application, to check the trips between home and an event, or between two events, the user can press the "Trips" button on the bottom left side of the screen (the light blue one with a plane on it) and the main screen will change to show the trips and all the confirmed events will become black and white.

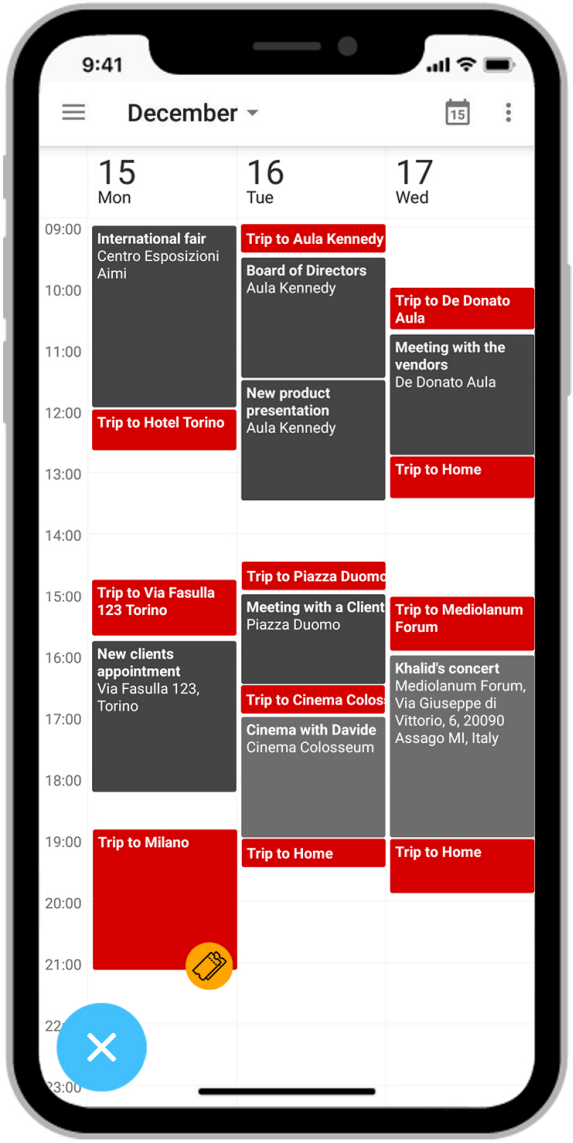

Figure 3.5: Mockup of the screen that contains all the trips computed by the application for the user

When the user taps on a trip he can check the details of it. If a trip presents an orange icon with some tickets on it, the user have to buy some tickets to complete the trip.

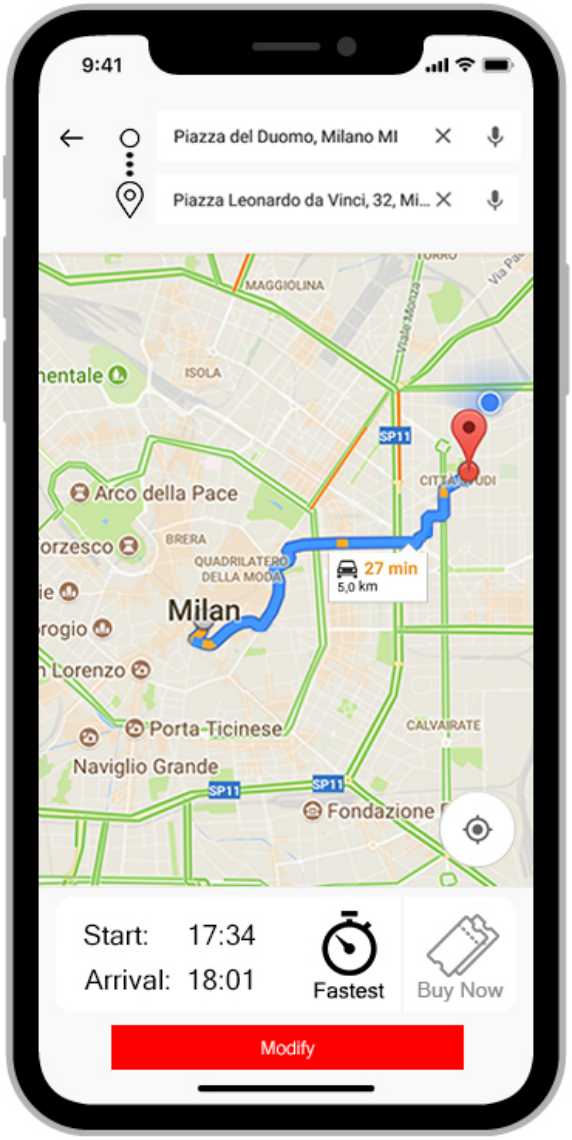

Figure 3.6: Mockup of the screen that shows the details of a trip

This screen shows the trip details: the route, the ETA, the time of departure, the time of arrival, the type of trip selected. If some tickets are needed the "Buy Now" button with the tickets on it is grey, but black and it will be clickable. The "Buy Now" is juxtaposed with the "Sharing" button if the trip include some section in which car sharing or bike sharing are used; clicking on the "Bike sharing" or "Car sharing" button will show up a screen with the cars and bike around the user of the various local sharing service.

If an user wants to modify a trip, he just have to press "Modify" button and a screen will appear when he can change the trip options.

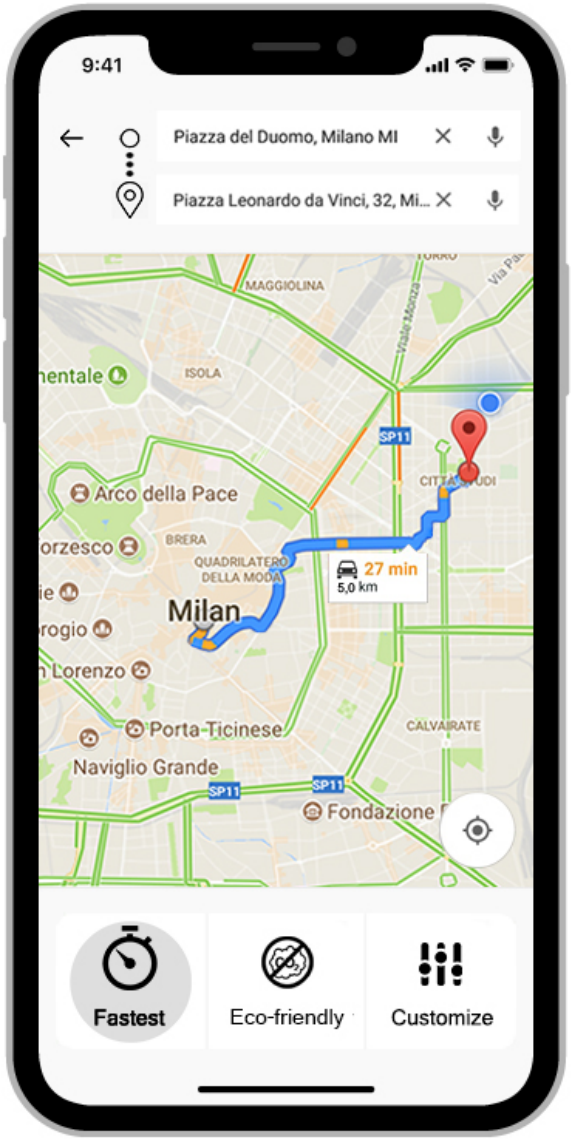

Figure 3.7: Mockup of the screen where the user can change the trip options

The system select by default the fastest trip options, but the user can change them. It is also possible to choose a trip that minimize the carbon footprint.

When the application need to communicate messages to the user, while

it's open and the screen is turned on, it will show a pop up, otherwise it will send a push notification to the user.

3.B.2 Hardware interfaces

The main hardware interface used by the system is the GPS, it's used in order to correctly position the user in the map. Since the application uses the internet connection, all the hardware required to connect to the internet will be hardware interface for the system.

3.B.3 Software interfaces

The mobile application is made up using mainly two Google APIs: Google Maps and Google Calendar. It relies also on other APIs: one for the weather forecast and one for each car sharing, bike sharing service and for the public mobility. It is developed for the use on the two most common mobile operating systems: Android and iOS.

3.B.4 Communication interfaces

The application communicates with the server using Java Remote Method Invocation (RMI).

3.C Performance Requirements

1. The system must support 500 contemporary requests;
2. 90% of requests must be processed in less than 3 seconds;
3. 100% of requests must be processed in less than 10 seconds;
4. There's no limit on the number of users;
5. Trips solutions are updated daily until the day of the trip, then every hour until half an hour before the trip, finally every minute until the trip begins;
6. The system will compute continuously all the travel options for the different users, but must find a free time spot to answer to new requests.

3.D Design Constraints

3.D.1 Standards compliance

This document follows the IEEE Standard 830-1998 [7] for the format of Software Requirements specifications.

3.D.2 Hardware limitations

The user will need a smartphone with at least:

- 3G connection
- GPS
- Enough storage on smartphone

3.E Regulatory policy

The System asks the User for the permission to acquire, store and use his personal data, and informs him that won't take any responsibility for a use of it that doesn't complies with the local laws and policies, by means of the User agreement's acceptance. The System under request of the User must delete all his personal data.

3.F Software System Attributes

3.F.1 Reliability

The system shall have an availability of 99.95% ("three and a half nines"). It means that the application will have at most a downtime per year of 4.38 hours.

3.F.2 Availability

The system will run 24/7 in order to make the user manage his events and the trips between them whenever he wants. Furthermore any kind of update must not stop the normal running of operations.

3.F.3 Security

The main security features of the application are:

- All the meetings and the trips must be kept private.
- Enable SSL/TLS encryption protocol for Client-Server communication to protect from internal and external threats, depending on user's network configuration. Enabling SSL/TLS ensures the confidentiality, authentication, and integrity of session data. Authentication is required for both server and client
- Insecure communication channels will lead to a refuse for the client's request.
- Passwords must be encrypted, hashed and salted before they could be stored in the databases.
- Users can access in reading or in writing on only a limited set of data.

3.F.4 Maintainability

The application code will be well documented to let future developers understand how it work and to make them able to modify it.

3.F.5 Portability

The application will be available for the two most common mobile operating systems: Android and iOS.

Chapter 4

Scenarios

The following tables present some examples of usage of this application:

Scenario 1: First use of the application
Giovanni Giorgio is a famous Italian musician, who needs to travel a lot around the big city where he lives. He has many daily appointments, so he wants to optimize his displacements, and, on suggestion of his best friend Francesco, he downloads on his brand-new smartphone the Travlendar+ application. As he opens it he is asked to log in or register to the service, so he proceeds to register filling up the form with his email address, password, personal data and to accept the user agreement on personal information. He then gets a warm welcome message, explaining him that only this time a little set up is needed. The following application page opens letting him set up his favourite locations, so he enters his house, workplace and children's school; than he is asked to set up his break times schedule, thus he chooses that from Monday to Friday between 12.30 AM and 2 PM Travlendar+ will keep half an hour free from trips for him to have lunch; next he can customize the time of the day after which using the bike or walking will not be considered anymore by the application, and he chooses 8 PM. He is also asked if he desire to log into his Google account to synchronize the calendar and maps, and he is glad to do that. After he can select his means of travel: he has a car, uses train and frequently gets public transports to move in the city centre, therefore he inserts his season pass; at last he set up to 1,5 km the maximum distance he is willing to walk. Done all of that he notices that he received an email from the Travlendar+ staff asking to click a link to confirm his email, so he does that and begins to use the application.

Scenario 2: Standard usage (default options, notification, map consultation)
For everyday he needs to go to work in his recording studio, he opens the newly acquired application to set up this event. He goes there from Monday to Friday, beginning at 8 AM and finishing at 12.30 AM so he sets it as a daily recurrent event, excluding Saturday and Sunday, he adds the timetables and chooses location from the favourites. The next day, while he is having breakfast at 7.13 AM, his smartphone beeps in a welcoming way, showing a notice from Travlendar+ that he needs to leave his house in half an hour to reach his workplace in time using the public transports. Thus, he opens the application clicking on the notification and he is presented the trip details screen, showing him on the map the road he needs to take to catch the bus to go to work and that the displacement will take an ETA of 12 minutes in total, from 7.43 AM to 7.55 AM. Giovanni Giorgio, happy to see his new application working as intended - if not, better - leaves his house at the suggested time and gets to work on time. To get home after work hours Travlendar+ suggests that he goes again by public transportation, knowing he didn't bring his car to work.

Scenario 3: Minimize carbon footprint
Some days later our Giovanni Giorgio in his free time is exploring the Travlendar+ application, so he presses on the travels button on the main screen and the calendar view transforms, making his events black and white. Now, before and after his appointments, the trips appeared on the calendar, showing their timetables and an indication of the suggested means of transport. Gazed in curiosity, he presses on the trip to the dentist he added some time before, which says he should go to the by car. The expanded screen containing the path on the map and trip details - labelled as fastest - opens with a soft animation. On the bottom he sees a modify button under the trip details, he presses it and three options show up: fastest, eco-friendly and customized. So, he decides to try an eco-friendly solution because he is not in a hurry and his dentist in not so far away from his house. Therefore, the application suggests going there by foot showing him a new path crossing the park and he gladly keep it as it is now.

Scenario 4: Passengers
Sometimes Giovanni Giorgio's wife needs his to go take the children at school, so he creates the event and then he selects the forward trip and changes the passengers' indicator from the default one to three. In this way Travlendar+ will know that he is going by car or by public transportation. The children will have a nice trip this time too!

Scenario 5: Usage of sharing service
Giovanni Giorgio come to know about an innovative bike sharing service in his city and decides to try it. After enrolling using the dedicated application of the service, he opens Travlendar+ and in the settings, he adds the sharing service to his means of transport selecting it from a list. The other application opens, asking his permission to give information to Travlendar+; he accepts and goes back to Travlendar+, which will now suggest bike trips locating the nearest bike of the sharing service and use this information to calculate the estimated time of leaving from home.

Scenario 6: External event causing exception and consequent hand customized trip
On an unlucky day Giovanni Giorgio is going to have dinner with some friends and colleagues of the music industry in the city centre, so Travlendar+ suggests him to take his car (obviously not by bike because it is past 8PM). Unexpectedly his car breaks down halfway, so he grabs his smartphone to quickly open the trip and go to the customized option. It opens a menu letting him choose which means of transport he prefers instead of the car; he selects public transport and the application immediately updates the trip using his position. In the end he gets to his dinner on time using the bus, not losing his fame of being a timely man.

Scenario 7: Lunch break and external trip-changing factors
On a rainy day Giovanni Giorgio is full of appointments: he must go to work until 12.30 AM and at 2 PM his fans wait for him at a store to have some copies of his last EP signed. He will not be suggested going home to have lunch because there is not enough time to travel back and eat, additionally to the next appointment trip time. Travlendar+ knows it is not suitable to go by bike when it's raining, thus it suggests him to go by car. However, 20 minutes before he leaves Travlendar+ sends him another notification that due to the intense traffic, he should instead leave in twelve minutes taking the subway to get to work in time. Happy of the close call, Giovanni Giorgio arrives at work on time thanks to Travlendar+. After work the application suggests him to leave immediately for his next appointment location because rain stopped but the weather is going to worsen in an hour; then he will have time to eat from 1.30 PM to 2 PM. Obviously the application knows that until he does not come back home he cannot begin a trip with a personal means of transport, will not suggest going to the next event by car unless he come by car to an appointment. Our man follows the instructions, and everything goes as planned; by now he is in love with his Travlendar+ application.

Scenario 8: Buy ticket
Our artist needs to go in a county town near his city to see a house he wants to buy, so he trusts Travlendar+ to arrange the trip for him. The application suggests using the train and the trip screen contains a button indicating the possibility of buying the ticket. He presses it and the application opens a built-in browser that redirects to the train company page selling tickets, so he proceeds buying the ticket with his credit card. The browser closes, and the check-out screen compares. Everything is ready for the trip!

Chapter 5

UML Models

In the following pages are present some models that describe the system and its functionalities.

5.A Use cases

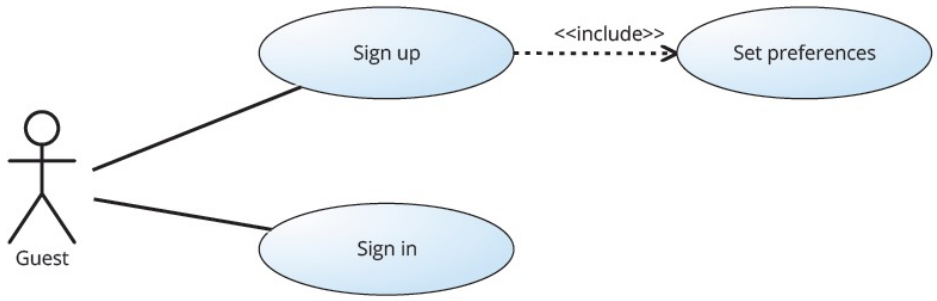

Figure 5.1: Guest's use case diagram

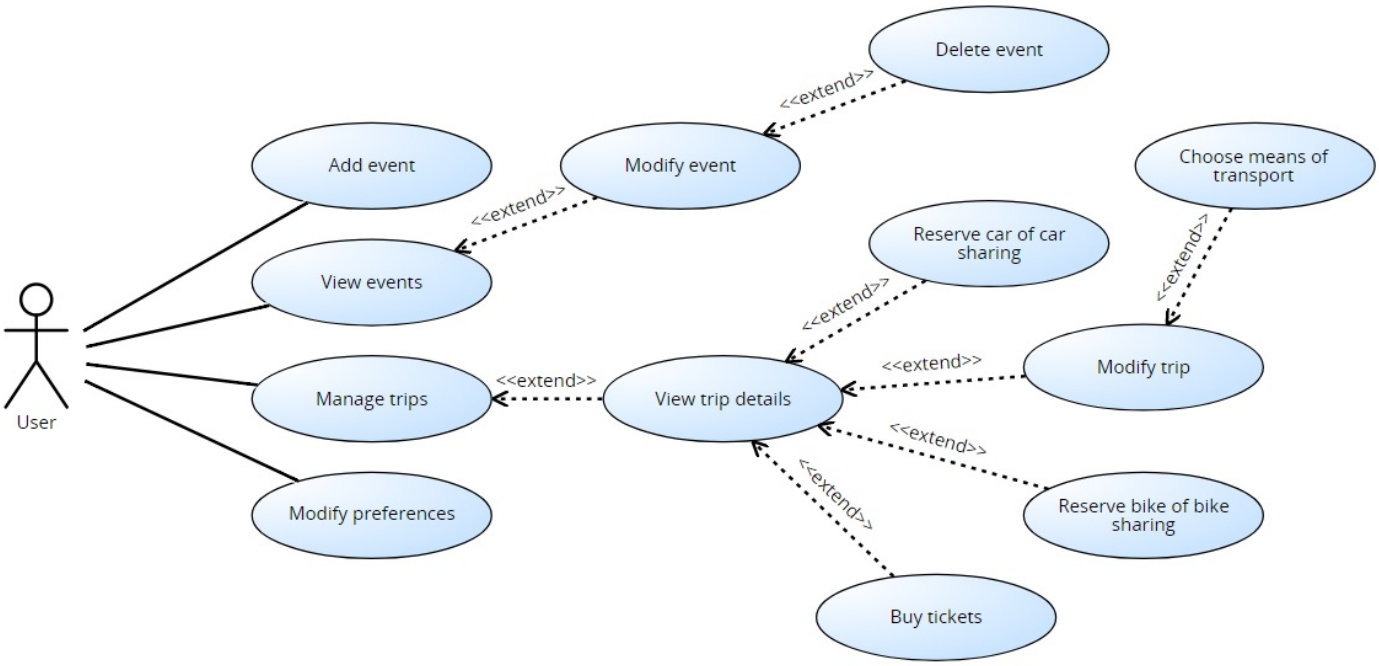

Figure 5.2: User's use case diagram

5.B Class diagram

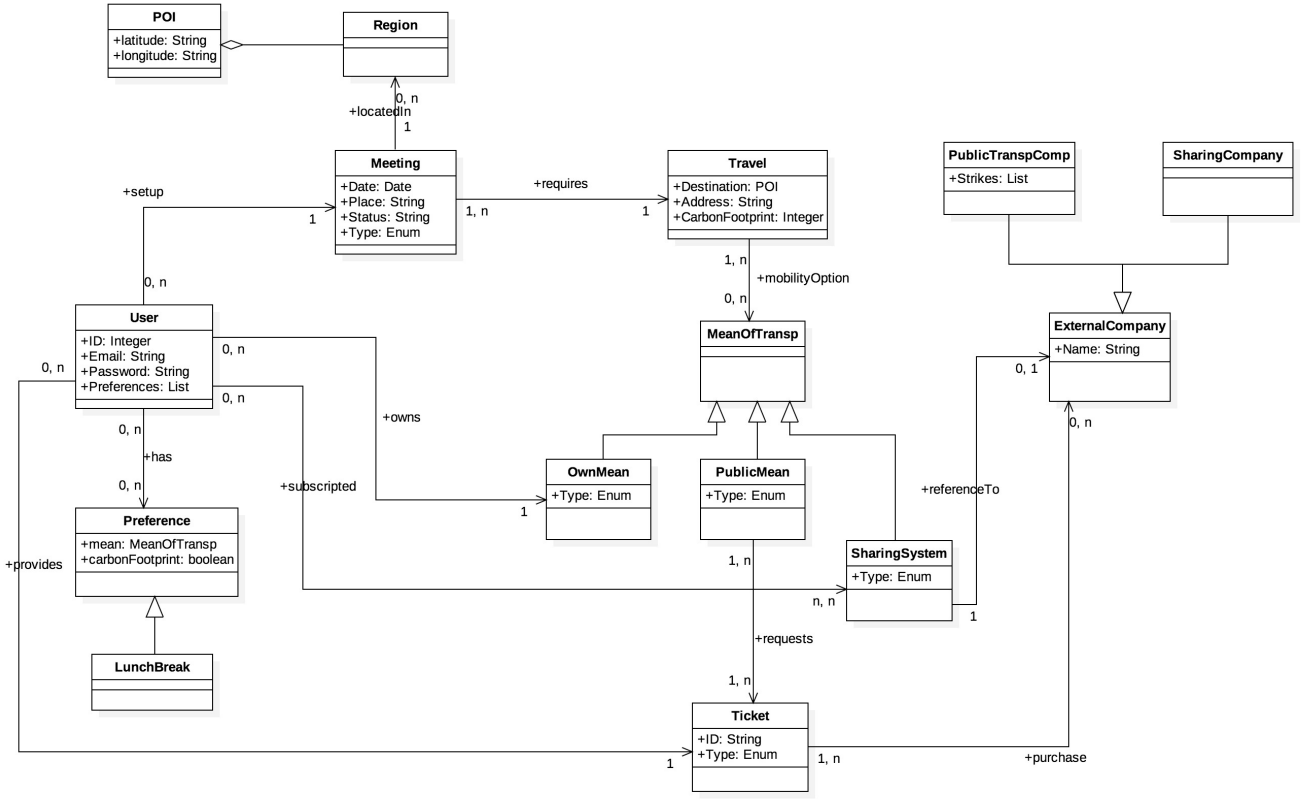

Figure 5.3: Class diagram of the application

5.C Use case description

In this section some use cases will be described. These use cases can be derived from the scenarios and the use case diagram.

Name	Sign up
Primary Actor	Guest
Preconditions	The guest wants to register to "Travlendar+"
Postconditions	Guest's informations are stored in the "Travlendar+" server and locally on the device. The Guest can sign in to use the application, becoming a user
Flow of events	1. The guest opens the application for the first time 2. The system shows the Login screen to the guest 3. The Guest clicks on "Sign Up" 4. The System shows to the Guest the Registration page 5. The guest inserts his personal details (name, surname, ...) and his trip preferences (important addresses, owned car, season ticket, ...) 6. The guest reads and accepts the user agreement 7. The guest taps on "Confirm" 8. The system check the correctness of the data and sends an email and a sms with a verification link 9. The guest confirms his registration clicking on one of the verification links 10. The guest is now registered and becomes a User of "Travlendar+" 11. The system shows to the user the Main screen (Calendar) of the application
Exceptions	1. One or more fields of the Registration page are not well formed 2. Username is already in use 3. Email is already in use 4. The verification link is expired (after 24 hours)

Table 5.1: Use case for sign up

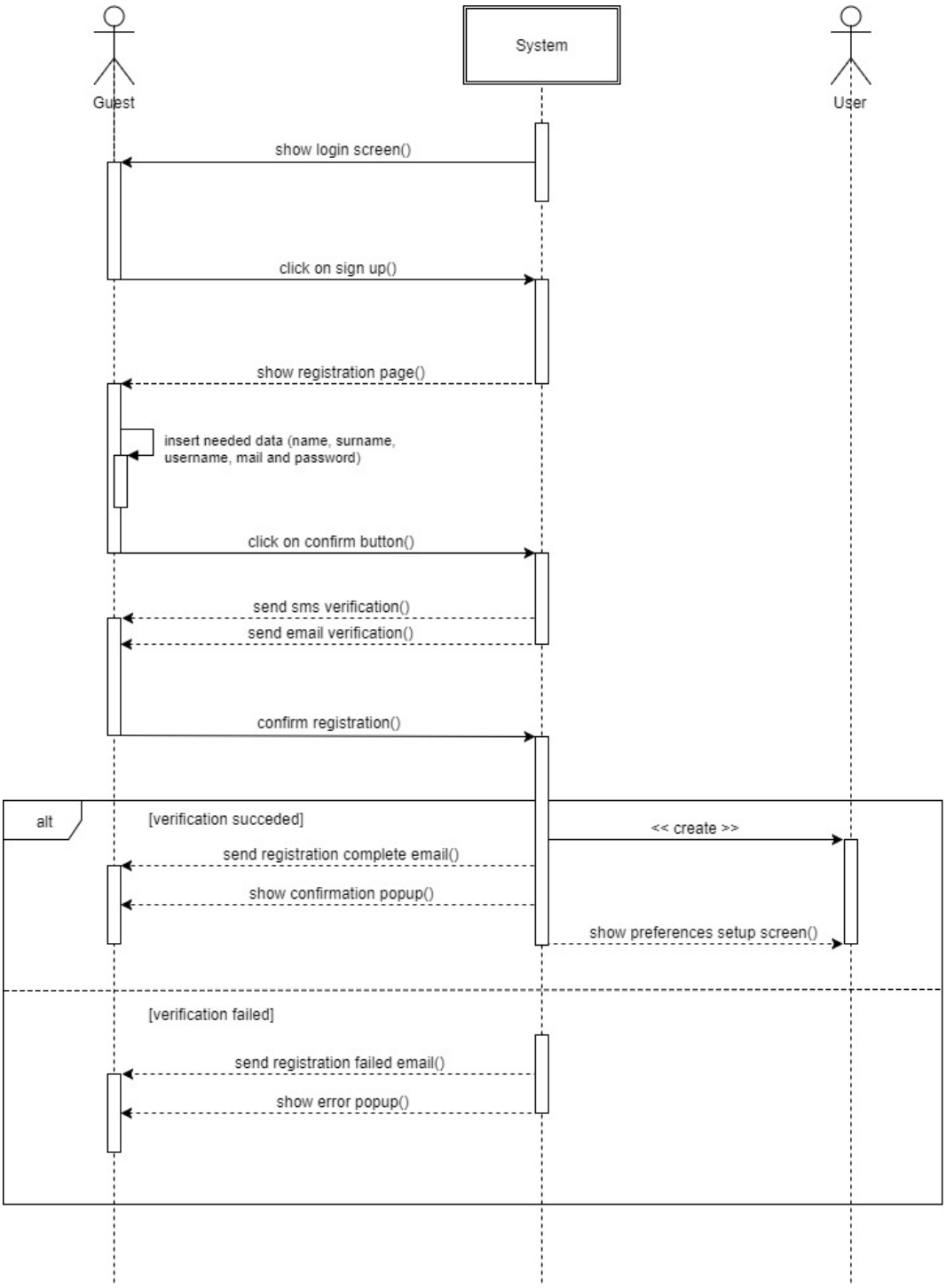

Figure 5.4: Sequence diagram of sign up use case

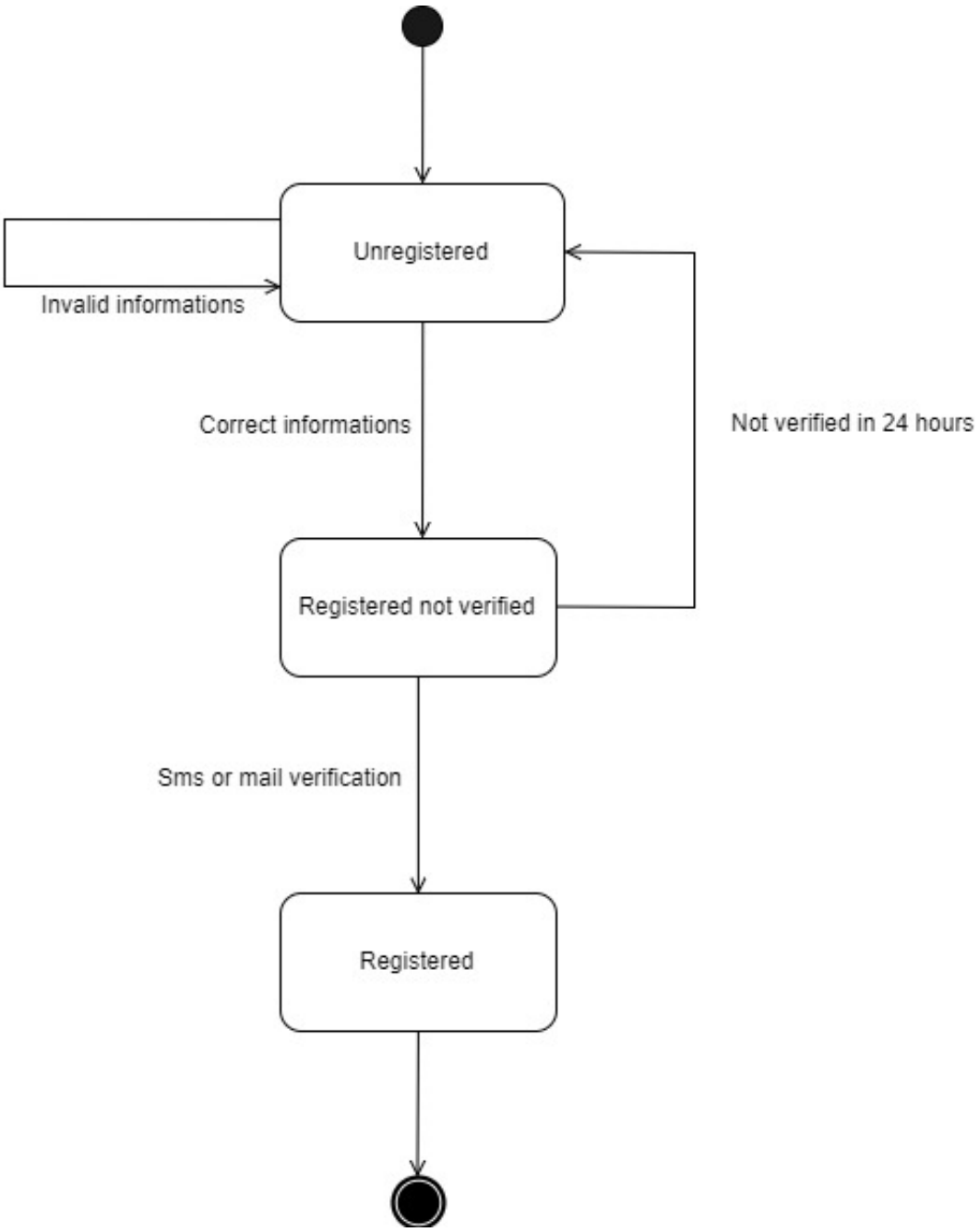

Figure 5.5: Statechart of sign up process

Name	Sign in
Primary Actor	Guest
Preconditions	The guest wants to sign in to use “Travlendar+”. He is actually an user of the application because he has a valid account for it
Postconditions	The guest is now logged into the system becoming an user
Flow of events	1. The guest opens the application 2. The system shows to the guest the Login screen 3. The guest inserts account credentials (username and password) 4. The guest taps on “Sign In” 5. The system checks if the credentials are present in the database 6. The guest is now logged and becomes an user 7. The system shows to the user the Main screen of the application
Exceptions	1. One or more fields of the Login page are not well formed 2. Username is not present in the database 3. The password associated to the username is incorrect

Table 5.2: Use case for sign in

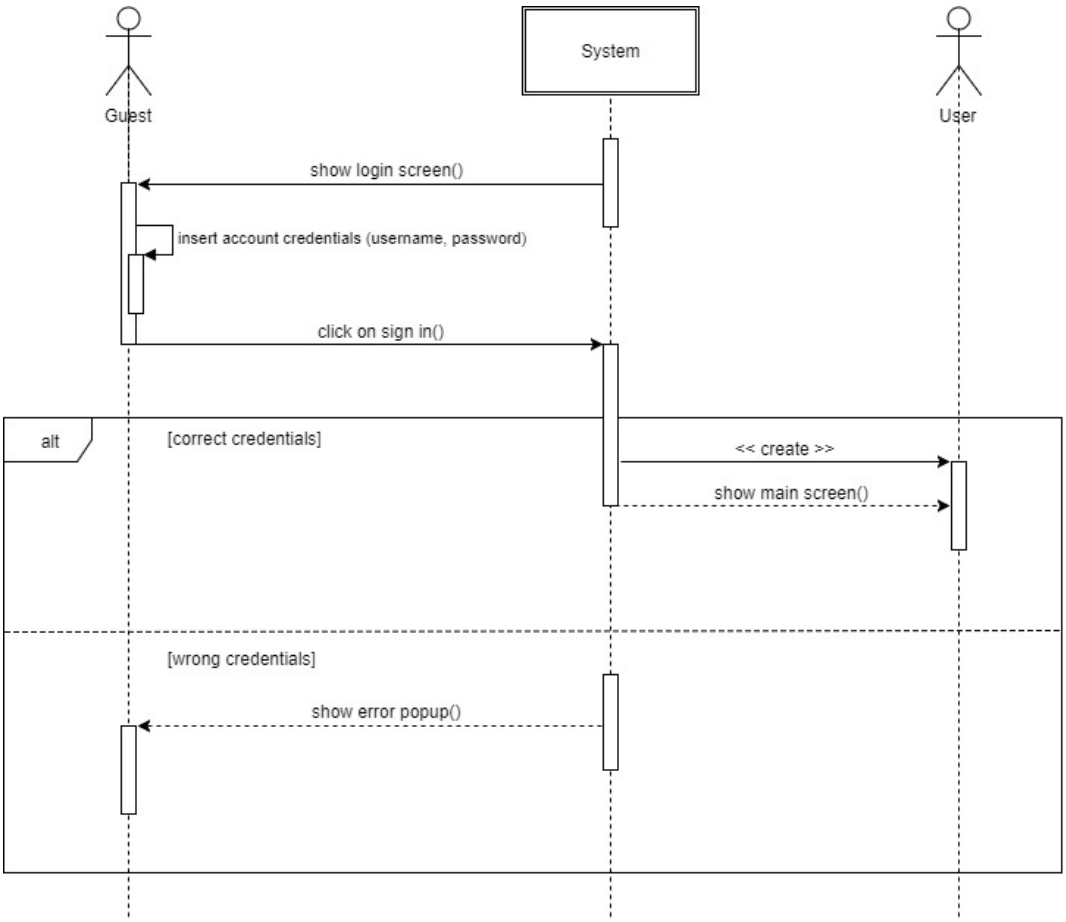

Figure 5.6: Sequence diagram of sign in use case

Name	Add event to calendar
Primary Actor	User
Preconditions	The user wants to add an event to the calendar of his application. He is already logged in into the service and the system is showing the calendar
Postconditions	The event is added to the calendar and stored in the database, the system computes the round trip from the location of the event and shows again the calendar to the user
Flow of events	1. The user taps on "Add Event" (the red button with a white cross) 2. The system shows the Add Event screen 3. The user inserts the details in the requested fields 4. The user clicks on "Confirm" 5. The system adds the event to the calendar and shows the updated Main screen to the user
Exceptions	1. The event overlaps with another one 2. The event overlaps with user's lunch 3. One or more field are not well formed 4. The user cannot arrive in time for the event

Table 5.3: Use case for add event

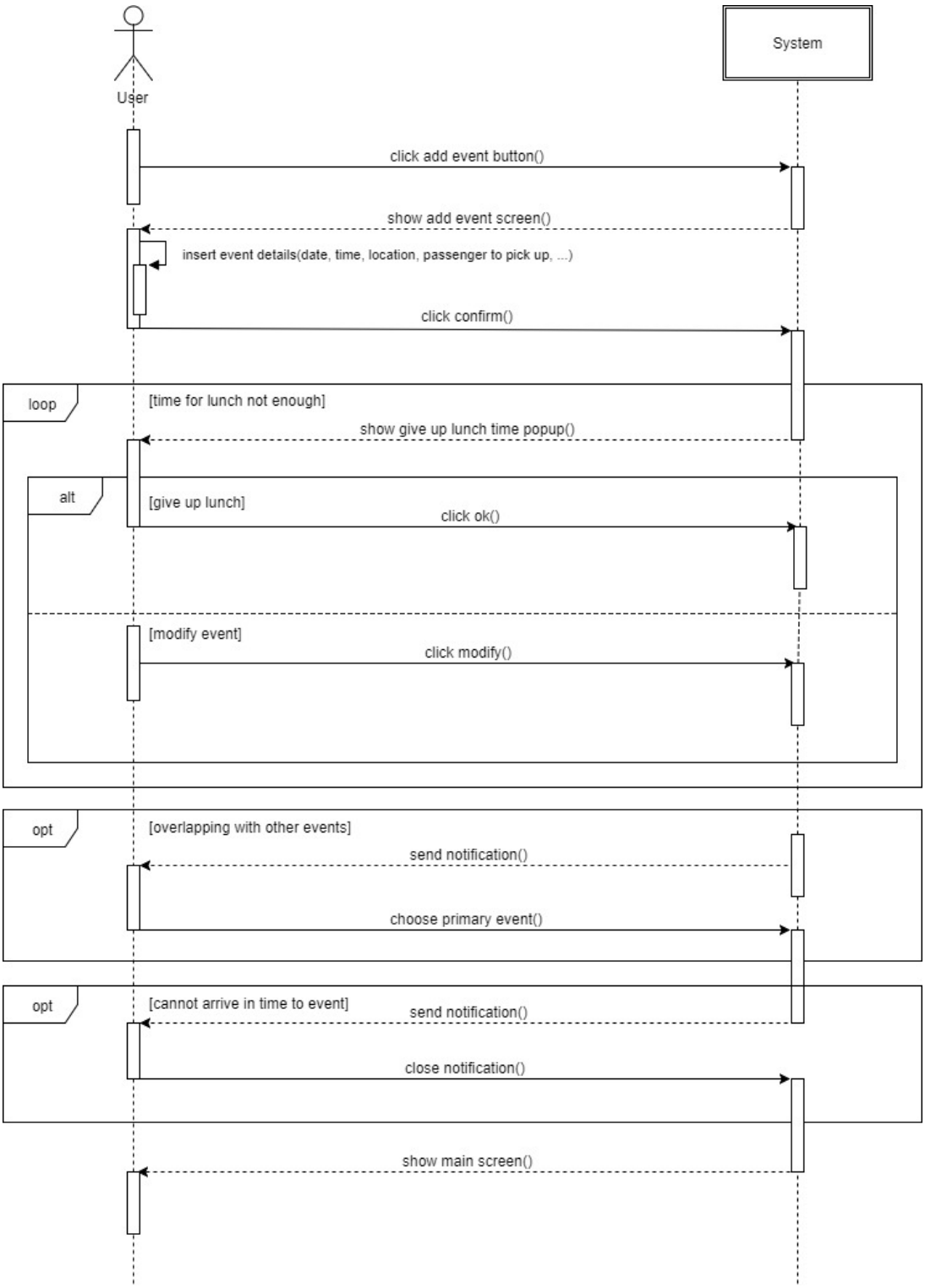

Figure 5.7: Sequence diagram of add event use case

Name	Delete event from calendar
Primary Actor	User
Preconditions	The user wants to delete an event from the calendar of his application. He is already logged in into the service and the system is showing the calendar
Postconditions	The event is deleted from the calendar and from the database, the system shows again the calendar to the user
Flow of events	1. The user selects the event he wants to delete 2. The system shows the Event Details screen 3. The user taps on "Modify Event" 4. The system shows Event Edit screen 5. The user clicks on "Delete Event" 6. The system deletes the event from the calendar and from the database, and shows the updated calendar to the user
Exceptions	1. The event overlaps with at least other two events and it is primary

Table 5.4: Use case for delete event

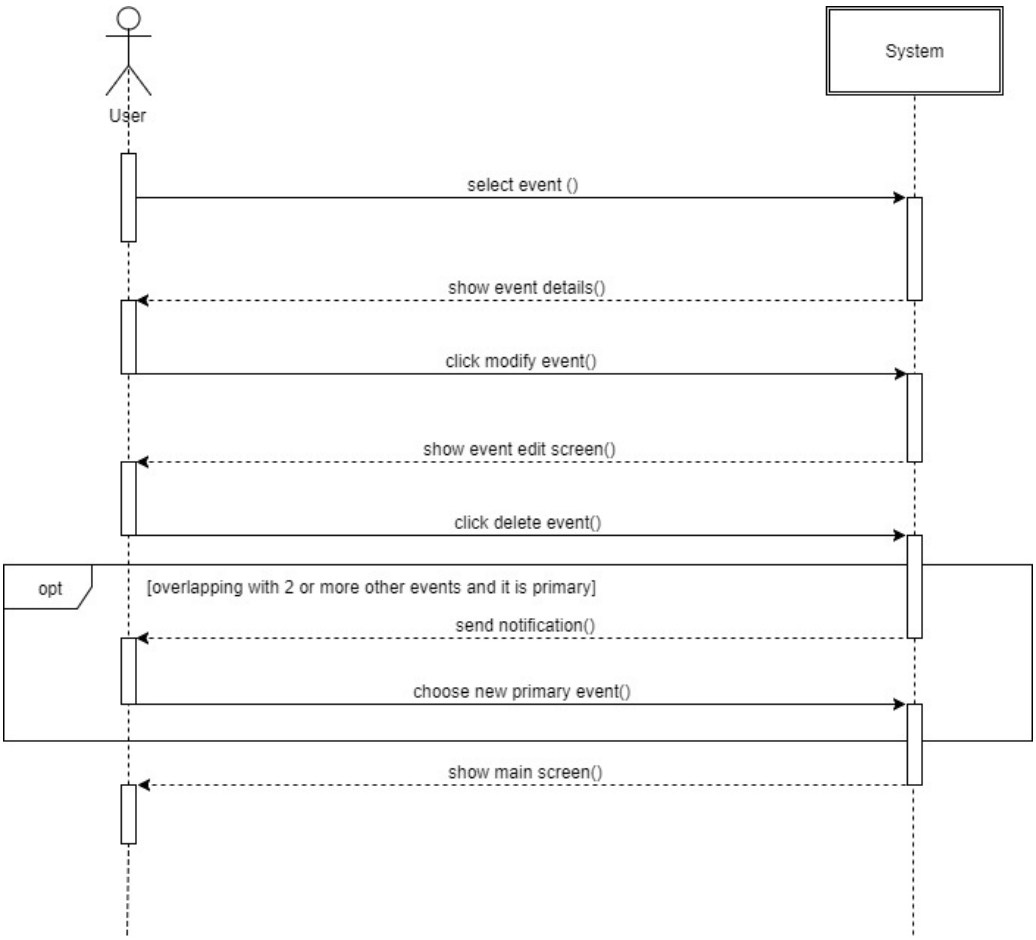

Figure 5.8: Sequence diagram of delete event use case

Name	Arrange trip
Primary Actor	User
Preconditions	The user wants to buy the tickets to take a trip. He is already logged in into the service and the system is showing the calendar
Postconditions	The user receives the tickets
Flow of events	1. The user clicks on “Trips” (the button with the airplane) 2. The system shows the Trips screen 3. The user chooses the event he wants to buy the tickets for 4. The system shows the Trip details screen 5. The user clicks “Buy Now” 6. The system opens a link to buy the ticket and shows the checkout page to the user for every ticket purchasable online 7. The user pays and confirms for all the tickets 8. The system calls the External services that send the tickets to the user 9. The user communicate to the system that the tickets are bought
Alternative flow	1. The same operations as above until point 5 2. The system calls the External services applications that show the checkout screen to the user, for every ticket only purchasable through the proprietary application 3. The user pays and confirms for all the tickets 4. The External services applications send the tickets to user 5. The user communicate to the system that the tickets are bought
Exceptions	1. One or more payment failed 2. The user doesn’t confirm payment for one or more tickets

Table 5.5: Use case for arrange trip

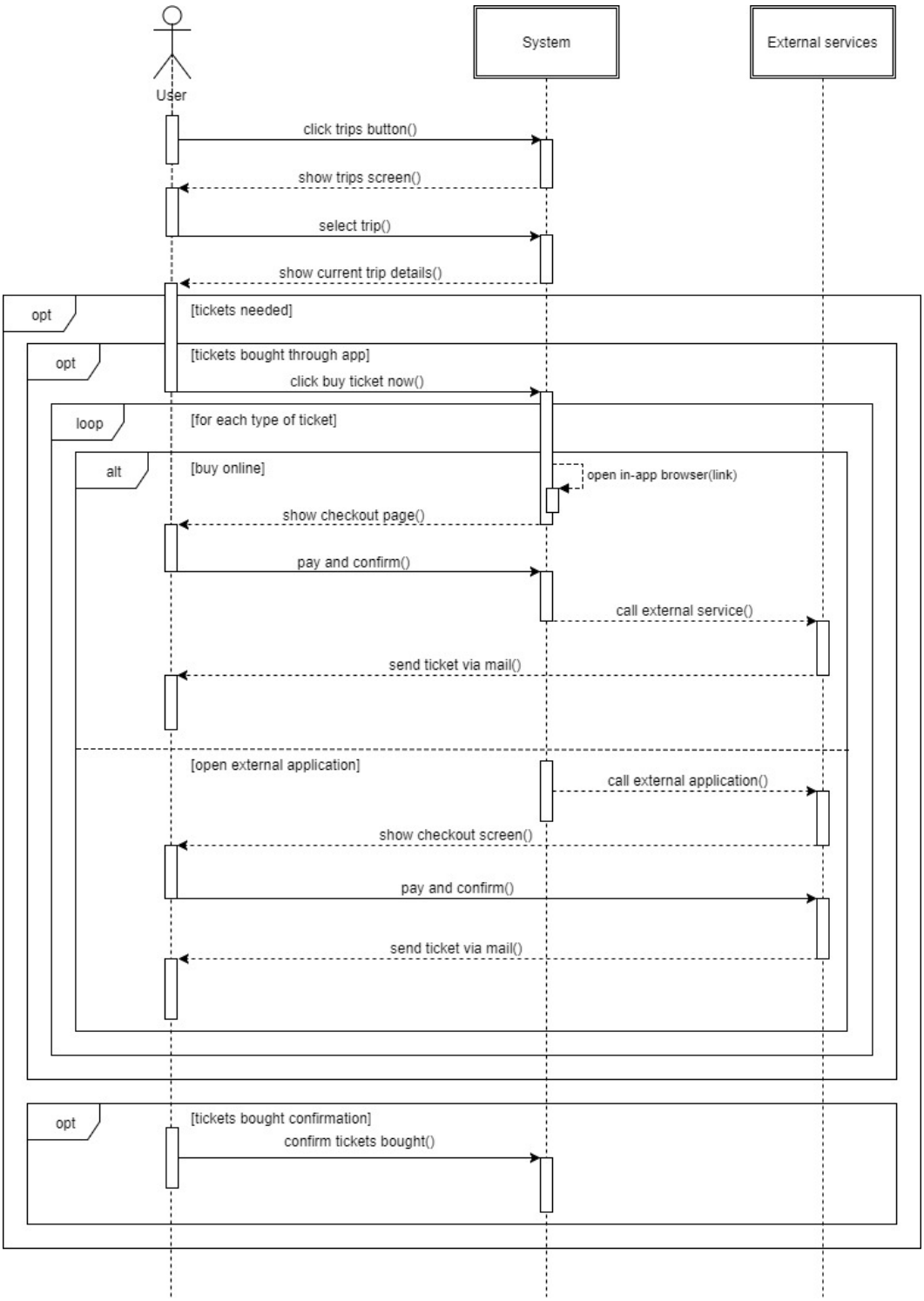

Figure 5.9: Sequence diagram of arrange trip

Name	Manage trip options (customized trip)
Primary Actor	User
Preconditions	The user wants to change the modality of one of his trips, choosing to customize it and changing the preferred means of transport. He is already logged in into the service and the system is showing the calendar
Postconditions	The trip is saved and changed as the user prefers
Flow of events	1. The user clicks on "Trips" (the button with the airplane) 2. The system shows the Trips screen 3. The user chooses the event he wants to modify 4. The system shows the Trip details screen 5. The user clicks "Modify" 6. The system shows Suggested Trip Choices screen 7. The user select "Customized" 8. The system calculates all available trip options and shows them to the user in the Trip Options screen 9. The user chooses new preferred means of transport 10. The system recalculate all the trip options with the User selection and show them to the user in the Trip Options screen 11. The user chooses his preferred trip option 12. The system shows the Trip details screen 13. The user selects back (the arrow icon) 14. The system shows the Main screen to the user
Exceptions	1. The user chooses to go only by foot but the distance to walk is wider than the one set in the preferences 2. The user chooses to go only by bike, but the time is not inside the interval chosen in the preferences

Table 5.6: Use case for manage trip options, in the case that the user chooses to customize the trip

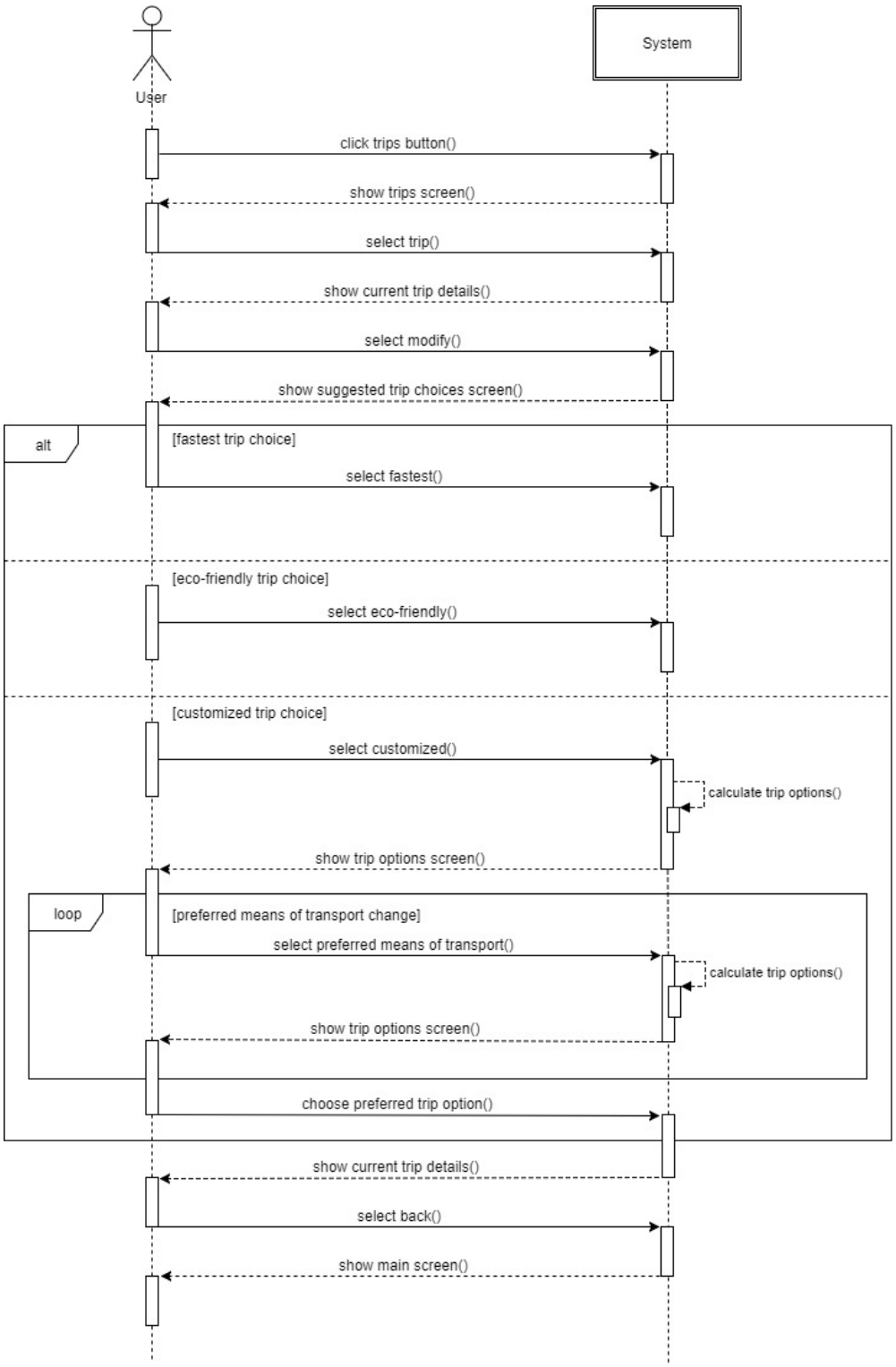

Figure 5.10: Sequence diagram of manage trip options

Name	Reserve a sharing means of transport
Primary Actor	User
Preconditions	The user wants to reserve a sharing means of transport. He is already logged in into the service and the system is showing the calendar
Postconditions	The user has reserved the means of transport chosen and receives a confirmation via mail
Flow of events	1. The user clicks on "Trips" (the button with the airplane) 2. The system shows the Trips screen 3. The user chooses the event that is active 4. The system shows the Trip details screen 5. The user clicks on the "Sharing" button 6. The system requests the position of the means of transport to all External Sharing Services and it shows to the user 7. The user chooses a means of transport 8. The system opens the application of the Sharing Service chosen 9. The user reserves the means of transport 10. The Sharing Service application sends a confirmation via mail and shows the means of transport reserved
Exceptions	1. The Sharing Service application is not installed on the device 2. The user doesn't have an account for the Sharing Service

Table 5.7: Use case for reserve a sharing means of transport

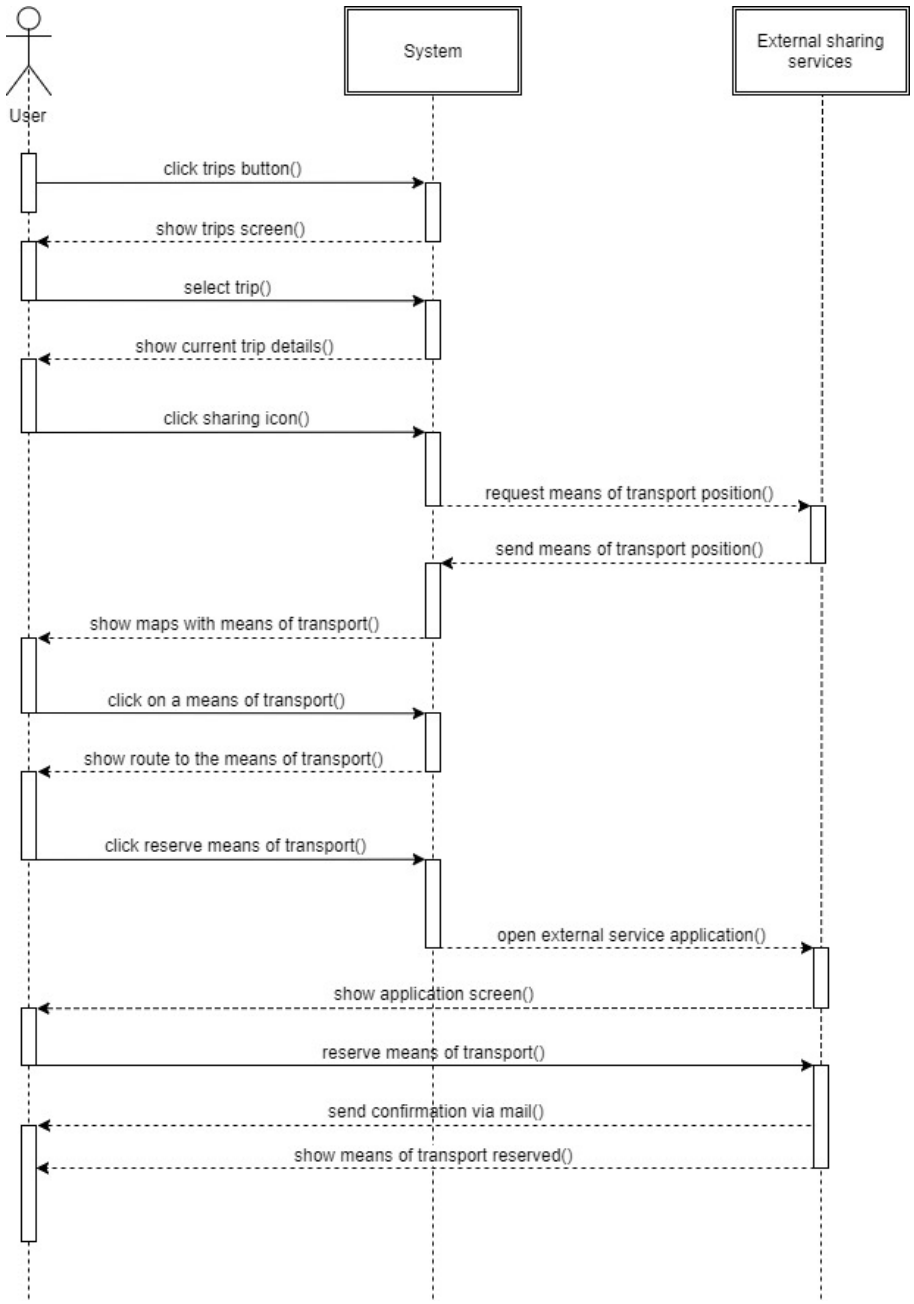

Figure 5.11: Sequence diagram of reserve a sharing means of transport

Name	Modify user preferences
Primary Actor	User
Preconditions	The user wants to change his preferences and the system is showing the main screen (calendar)
Postconditions	The new user preferences are saved
Flow of events	1. The user presses the the menu button 2. The system shows the menu sidebar 3. The user clicks on preferences menu 4. The system shows the preference menu 5. The user modify his preferences 6. The user clicks on confirm changes 7. The system shows the main screen
Exceptions	/

Table 5.8: Use case for modify user preferences

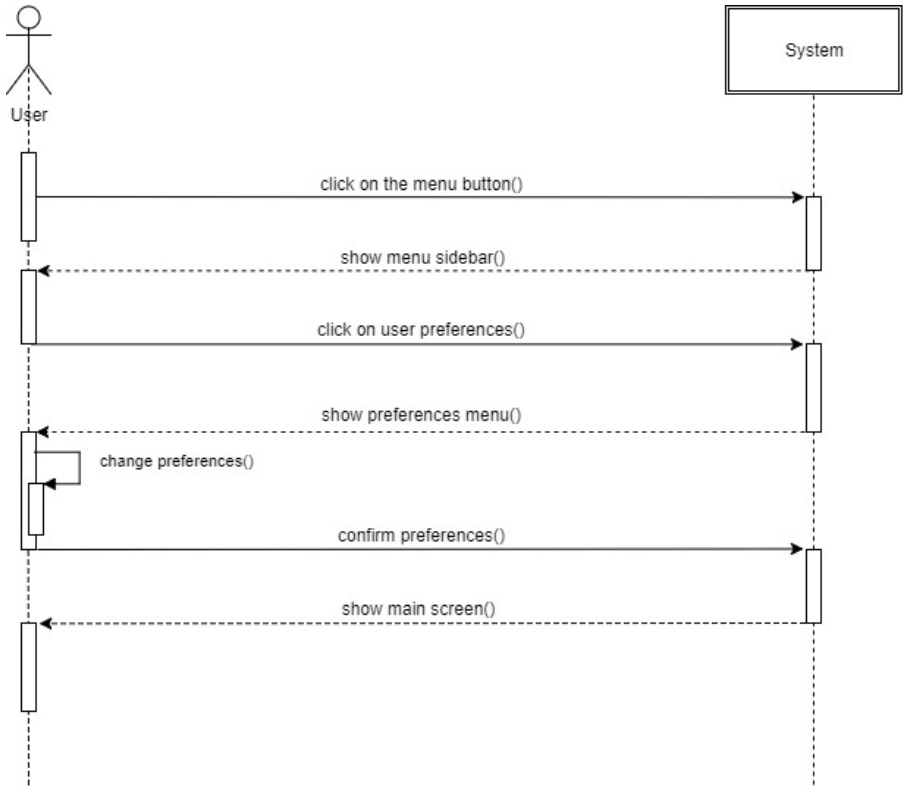

Figure 5.12: Sequence diagram of modify user preferences use case

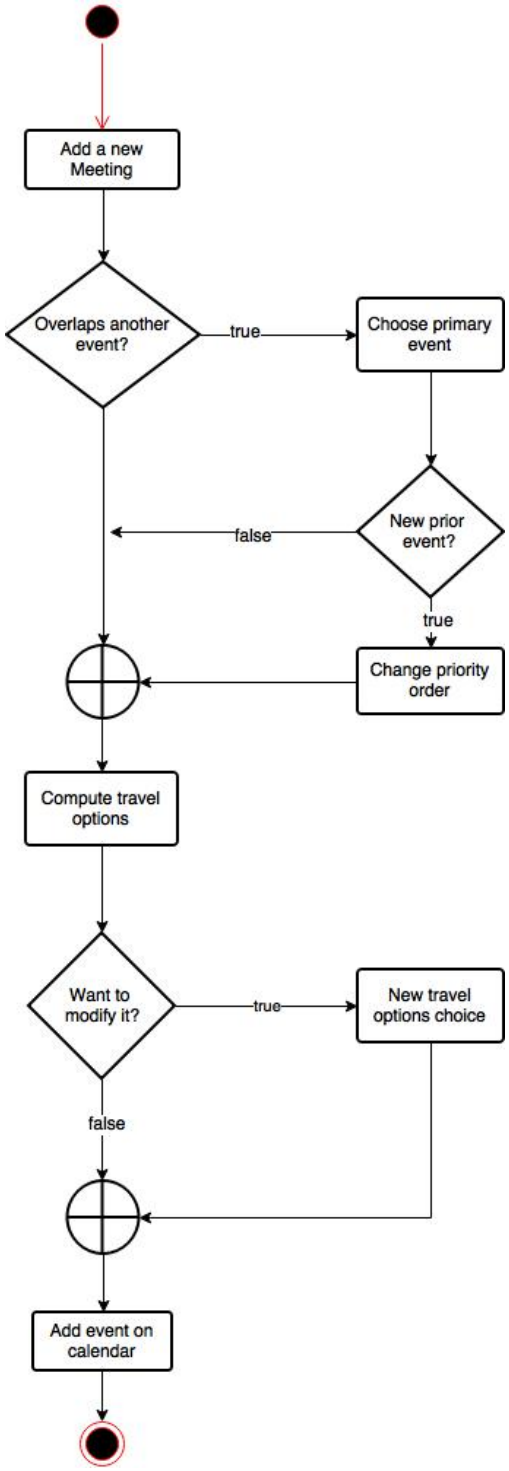

Figure 5.13: System's point of view activity diagram of an event addition and trip option choice

Chapter 6

Formal Analysis

6.A Modeling

```
open util/boolean

//Dates are expressed as the number of days from
   ↪ 01/01/2000 because that's for our
//needs the best way to manage them, since Alloy
   ↪ didn't have a date object

//********** SIGNATURES **********

sig User{
        email: Stringa,
        preferences: Preference, //one because the
           ↪ user can only have a set of
           ↪ preferences
        schedule: seq Meeting,
        owns: set OwnMean,
        subscribed: set SharingSystem,
        provides: set Ticket,
}{
        (#subscribed>0 or #owns>0 or #provides>0)
        #schedule>0
}

sig Stringa{}
```

```
22 //Set of preference in boolean that the user will
     ↪ compile when he registers on Travlendar+
   sig Preference{
24         carshare: Bool,
           bikeshare: Bool,
26         ownMean: Bool,
           publicMean: Bool,
28         walking: Bool,
           house: Bool,
30         work: Bool,
           minCarbon: Bool,
32 }{
           all p: Preference | one u: User | p in u.
             ↪ preferences
34 }

36 //Represents any type of meeting that the user can
     ↪ setup
   sig Meeting{
38         date: Int,
           startingTime: Int,
40         endingTime: Int,
           requires: set Travel,
42         locate: one Region,
   }{
44         date>0 ∧
           startingTime>0 ∧
46         endingTime>startingTime ∧
           #requires>0
48 }

50 sig Region{}

52 //Represents any type of travel and it's always
     ↪ associated to a meeting
   sig Travel{
54         date: Int,
           startingTime: Int,
56         endingTime: Int,
           mean: set MeanOfTransp,
58         needed: one Ticket,
```

```
        associated: one Meeting,
}{
        startingTime>0 ∧ date>0
        ∧ endingTime>startingTime ∧ endingTime <
            ↪ associated.startingTime
        ∧ date = associated.date
}

abstract sig MeanOfTransp{
        ticketNeeded: Bool,
}

sig OwnMean extends MeanOfTransp{
}{
        ticketNeeded=False
}

//Represents any type of public mean
sig PublicMean extends MeanOfTransp{
        need: one Ticket,
}{
        ticketNeeded=True
}

//Represents any type of sharing system
sig SharingSystem extends MeanOfTransp{}
{
        ticketNeeded=False
}

//Represents any type of ticket for public means
sig Ticket{
        seasonPass: Bool,
}

//********** FACTS **********

fact userHasTickets{
        some u: User | all t: Ticket | t in u.
            ↪ provides
}
```

```
98
   //Users will travel on mean they could take
100 fact noUnauthorizedMean{
        all u: User, t: Travel | t.mean in u.owns ∨
          ↪ t.mean in u.subscribed ∨
102     t.needed in u.provides
    }
104
    fact uniqueMail {
106     all u1, u2: User | (u1 ≠ u2) ⟹ u1.email ≠
          ↪ u2.email
    }
108
    //Preferences are connected to what the user owns or
      ↪  provides
110 fact consistentPreferences{
        all p: Preference | one u: User | p in u.
          ↪ preferences ∧ (p.carshare=True ⟹ #u.
          ↪ subscribed>0)
112     ∧ (p.ownMean=True ⟹ #u.owns>0)
        ∧ (p.bikeshare=True ⟹ #u.subscribed>0)
114     ∧ (p.publicMean=True ⟹ #u.provides>0)
    }
116
    //Meetings are connected to users
118 fact meetingOfAUser{
        all m: Meeting | one u: User | m in u.
          ↪ schedule.elems
120 }

122 //There are no duplicates in the sequence of
      ↪ meetings
    fact noDuplicates{
124     all u: User | not u.schedule.hasDups
    }
126
    //Travels associated to a user must start at
      ↪ different times
128 fact travelsStartAtDifferentTime{
        no disj t1, t2: Travel | one m: Meeting | m
          ↪ in t1.associated ∧ m in t2.associated
```

```
130         ∧ t1.date=t2.date ∧ t1.startingTime=t2.
               ↪ startingTime ∧ t1.endingTime=t2.
               ↪ endingTime
    }
132
    fact needTicketIfPublicMean{
134         all t: Travel | all m: PublicMean | #t.
               ↪ needed>0 <=> m.ticketNeeded=True
    }
136
    //Any Travel needs at least one mean of transport
138 fact AtLeastOneMeanForTravel {
            some m: MeanOfTransp | some tick:Ticket |
               ↪ all t: Travel |
140         m in t.mean ∨ tick in t.needed
    }
142
    //Travel must be compatible to what the user owns or
       ↪  provides
144 fact TravelCompatibleToUser{
            all t: Travel | all u: User | all i: u.
               ↪ schedule.inds  | t in u.schedule[i].
               ↪ requires ⟹
146         ((some m: t.mean |  m in u.schedule[i].
               ↪ requires.mean) ∨
            (some tick: Ticket | tick in u.schedule[i].
               ↪ requires.needed))
148 }

150 fact travelAssociatedToMeeting{
            all t: Travel | one  m: Meeting | t in m.
               ↪ requires
152 }

154 fact MeanExistsOnlyIfUsed{
            all m: MeanOfTransp | one t: Travel |
156         m in t.mean
    }
158
    fact TicketExistsOnlyIfUsed{
160         all tick: Ticket | one t: Travel | tick in t
```

```
            ↪ .needed
}

//********** PREDICATES **********

/**
* Precondition: not m in this.schedule.elems
*/
pred User.addMeeting[u: User, m: Meeting] {
this.schedule = this.schedule.add[m]
}

/**
* Precondition: not this.schedule.isEmpty
*/
pred User.deleteMeeting[u: User, m: Meeting] {
        m in this.schedule.elems ∧
        not this.schedule.hasDups ∧
        this.schedule.lastIdxOf[m]=0
        u.schedule = this.schedule.delete[this.
            ↪ schedule.idxOf[m]] ⟹
    not m in u.schedule.elems
}

pred show{
#User=1
#Meeting=2
}

//********** ASSERTIONS **********

assert addChangesSchedule {
        all u1, u2: User, m: Meeting | #u1.schedule
            ↪ = #u2.schedule ∧ (u1.addMeeting[u1, m
            ↪ ] ⟹
        #u2.schedule < #u1.schedule)
}

assert deleteInverseOfAdd {
        all u: User, m: Meeting, s: u.schedule | u.
            ↪ addMeeting[u, m] and u.deleteMeeting[u
```

```
            ↪ , m] ⟹
        s = u.schedule
198 }

200 assert oneTravelAtATime{
        no u: User | one disj t1, t2: Travel | all i
            ↪ : u.schedule.inds | t1 in u.schedule[
            ↪ i].requires
202     ∧ t2 in u.schedule[i].requires ∧ t1.date=t2.
            ↪ date
        ∧ (t1.endingTime>t2.startingTime ∨ t2.
            ↪ endingTime>t1.startingTime)
204 }

206 //********** RUNS AND CHECKS **********

208 run show for 5

210 check oneTravelAtATime

212 check addChangesSchedule

214 run deleteMeeting for 1

216 run addMeeting for 1

218 check deleteInverseOfAdd
```

```
Executing "Run show for 5"
  Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
  21161 vars. 1200 primary vars. 52263 clauses. 1228ms.
  Instance found. Predicate is consistent. 886ms.

Executing "Check oneTravelAtATime"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9323 vars. 582 primary vars. 23051 clauses. 279ms.
  No counterexample found. Assertion may be valid. 2ms.

Executing "Check addChangesSchedule"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9294 vars. 588 primary vars. 22848 clauses. 193ms.
  No counterexample found. Assertion may be valid. 73ms.

Executing "Run deleteMeeting for 1"
  Solver=sat4j Bitwidth=4 MaxSeq=1 SkolemDepth=1 Symmetry=20
  1643 vars. 151 primary vars. 4290 clauses. 80ms.
  Instance found. Predicate is consistent. 35ms.

Executing "Run addMeeting for 1"
  Solver=sat4j Bitwidth=4 MaxSeq=1 SkolemDepth=1 Symmetry=20
  1637 vars. 151 primary vars. 4282 clauses. 52ms.
  Instance found. Predicate is consistent. 38ms.

Executing "Check deleteInverseOfAdd"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  9134 vars. 597 primary vars. 22325 clauses. 157ms.
  No counterexample found. Assertion may be valid. 5ms.

6 commands were executed. The results are:
  #1: Instance found. show is consistent.
  #2: No counterexample found. oneTravelAtATime may be valid.
  #3: No counterexample found. addChangesSchedule may be valid.
  #4: Instance found. deleteMeeting is consistent.
  #5: Instance found. addMeeting is consistent.
  #6: No counterexample found. deleteInverseOfAdd may be valid.
```

Figure 6.1: Result of Alloy Analyzer

6.B World generated

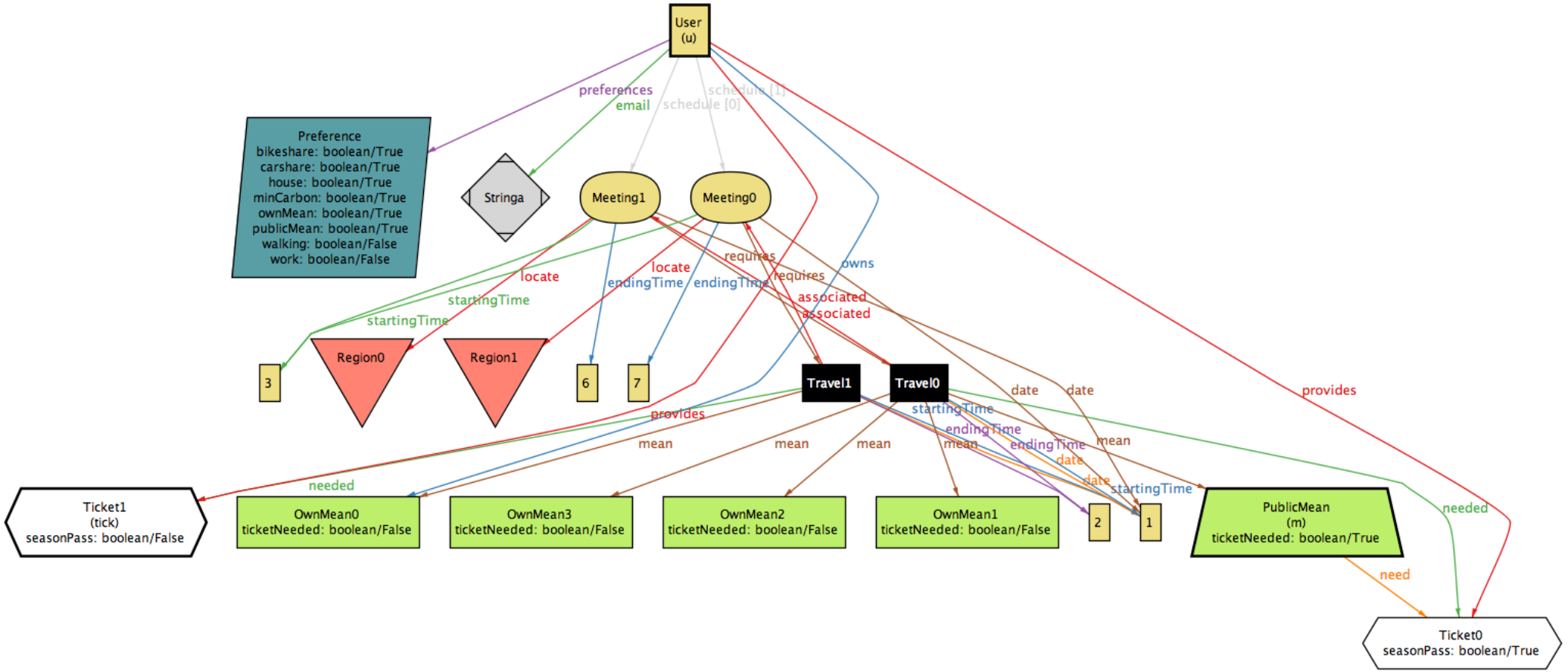

Figure 6.2: World generated

6.C Alloy vs UML

What we've modelled of the UML in alloy:
The goal of modelling in alloy is to describe some aspect of the system (but not the entire system), constrain it to exclude ill-formed examples and check properties about it. The section in UML concerning the external companies connected to Travlendar+, for instance the ones which can provide tickets or that can offer sharing systems, is not modelled in Alloy cause we assume that they will behave correctly in our system thanks to the agreements they took with us. We are also didn't model time because it's not useful in the model, but we will care about it in other portions of the project analysis. The time break will not be modelled and also the fact that the application will not advise trips by bike or foot on late hours.

Chapter 7

Effort Spent

Date	A. Aimi	R. Bigazzi	F. Collini
03/10/17	4h	5h	4h
05/10/17		1/2h	
06/10/17	2h		
07/10/17	1h	1h	1h
08/10/17			2h
09/10/17	4h	2h	
12/10/17	3h	3h	3h
14/10/17		1h	1h
15/10/17	1,5h	1/2h	
18/10/17	2h	2h	2h
19/10/17			3h
20/10/17	2h		
23/10/17	4h	4h	4h
24/10/17		2h	1h
25/10/17		3h	2h
26/10/17	1h	2h	
27/10/17	2,5h	1/2h	2,5h
28/10/17	5h	8h	6h
29/10/17	6h	6h	6h
Tot	38h	40,5h	37,5h

Table 7.1: Hours of work spent by the authors of the document