POLITECNICO
MILANO 1863

MSc Computer Science
and Engineering

Software Engineering 2
ACADEMIC YEAR 2017-2018

Requirements Analysis and Specification Document

894135
Franklin Onwu
franklinchinedu.onwu@mail.polimi.it

899318
Ivan Sanzeni
ivan.sanzeni@mail.polimi.it

884021
Matteo Vantadori
matteo.vantadori@mail.polimi.it

Related professor:
Prof. Matteo Giovanni Rossi

Release Date: October 29^{th} 2017
Version 2.0

Contents

1 Introduction

1.1 Purpose

Section [G.1] treats all the goals related to the creation and customization of a new event:

G.1.1 The user can schedule a new event adding name, time slot, location, type and description.

G.1.2 The user can modify the name of the event.

G.1.3 The user can modify the location of the event.

G.1.4 The user can modify the description of the event.

G.1.5 The user can modify the starting time of the event.

G.1.6 The user can modify the ending time of the event.

G.1.7 The user can modify the event type, from a work event to a personal event or vice versa.

G.1.8 The user can modify the description.

G.1.9 The user can choose how many minutes earlier arrive to the event location.

G.1.10 The user can delete an already existing event.

G.1.11 The user can see all the events he/she has already scheduled.

Section [G.2] treats all the goals related to the customization of the user preferences:

G.2.1 The user can find the quickest way to reach an event.

G.2.2 The user can find the cheapest way to reach an event.

G.2.3 The user can find the most ecological way to reach an event.

G.2.4 In adverse weather conditions, the user can find the best way to reach an event only using means that keep him/her protected.

G.2.5 The user can find the best way to reach an event imposing constraints to the time slots designated to any mean.

G.2.6 The user can find the best way to reach an event imposing constraints to the maximum distance covered by any mean.

G.2.7 The user can find the best way to reach an event with a maximum chosen budget.

G.2.8 The user can find the best way to reach an event only using chosen means.

Section [G.3] treats all the goals related to the customization of the user settings:

G.3.1 A user with disabilities can find the best way to reach an event according to his/her needs.

Section [G.4] treats all goals related to the purchase of *non-shared transports*:

G.4.1 The user can book a taxi.

G.4.2 The user can book a limousine.

Section [G.5] treats all the goals related to the purchase of *public transports*:

G.5.1 The user can buy a metro ticket.

G.5.2 The user can buy a bus ticket.

G.5.3 The user can buy a trolleybus ticket.

G.5.4 The user can buy a tram ticket.

G.5.5 user can buy a train ticket.

Section [G.6] treats all the goals related to the purchase of *shared transports*:

G.6.1 The user can take a bike from a bike sharing service.

G.6.2 The user can take a car from a car sharing service.

1.2 Scope

Travlendar+ is a calendar-based application designed to schelude any kind of event, supporting the user in reaching the location of the events all across Milan, combining different sort of means in relation to the user preferences.

The application is designed to match the user needs to personalize each event in every respect. So the user can easly customize each event assigning it a category and distinguishing it between work or personal reasons, deciding means and constraints to reach it and buying tickets or booking means in-app, if necessary.

The main application goal is to lead the user to handle each kind of event with *Travlendar+*: from a lunch with friends to a job interview, from an interesting expo to an out of town meeting.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Cheap = with this preference the application chooses the cheapest way to reach the location.

Eco = with this preference the application chooses the most ecological way to reach the location.

Flexible event = kind of event that provides calendar, reminder and street direction supports and can be overlapped with activities as long as exists a minimum amount of time fixed by the user.

Lasting event = kind of event that provides calendar, reminder and street direction supports and can be overlapped with activities.

Non-shared transports = limousine, taxi.

Not wet = with this preferences the application chooses only means that keeps the user out of adverse weather conditions to reach the location.

Personal event = the user specifies that the event has personal purposes.

Public transports = bus, metro, train, tram, trolleybus.

Quick = with this preference the application chooses the quickest way to reach the location.

Shared transports = bike sharing, car sharing.

Standard event = kind of event that provides calendar, reminder and street direction supports and cannot be overlapped with other activities.

Transfer event = kind of event that provides calendar and reminder supports and cannot be overlapped with other activities. It is used for events that take place outside Milan.

Travlendar+ = the name of the application.

Travlender = a registered and logged user of Travlendar+.

Work event = the user specifies that the event has work purposes.

1.3.2 Acronyms

API = Application Programming Interface.

GPS = Global Positioning System.

MMS = Mapping Managing System.

RASD = Requirements Analysis and Specification Document.

TMS = Transporting Managing System.

1.3.3 Abbreviations

A.n = Application assumption number *n*.

G.n.m = Goal number *m* in section *n*.

D.n = Domain assumption number *n*.

R.n.m = Requirement number *m* in section *n*.

T.n = Text assumption number *n*.

1.4 Revision history

29th October 2017
Version 1.0 - Document delivery.

7th November 2017
Version 1.1 - Improved the Alloy code and view following the *alloy-style.sty* package installation, added one more scenario, fixed some typing errors.

8th December 2017
Version 2.0 - Further improved the Alloy code, corrected the goals and the requirements, updated the use case tables, added a caption to each figure.

1.5 Reference documents

https://standards.ieee.org/findstds/standard
IEEE standard for requirements documents.

https://developers.google.com/maps
Reference point for the third-party *MMS* considered in this project.

https://citymapper.com/milano
Reference point for the third-party *TMS* considered in this project.

RASD Sample from A.Y. 2015-2016.pdf
First RASD document example from Software Engineering 2 directory, on BEEP.

RASD Sample from A.Y. 2016-2017.pdf
Second RASD document example from Software Engineering 2 directory, on BEEP.

1.6 Document structure

In the following parts we will introduce the application that allows the user to reach the 1.1 section goals, in order to satisfy in 1.2 section problem. The document is subdivided in other four parts, besides the introduction:

Overall description
A general description of *Travlendar+*, that includes a list of the external system interfaces, an explanation of the major system functions, a description of user characteristics in detail and all our assumptions and constraints during the app creation.

Specific requirements
A detailed description of all the *Travlendar+* requirements according to the IEEE standard: from the external interface to the functional requirements, from performance requirements to the design constraints, from the software system attributes to any other requirement.

Formal analysis using alloy
The complete description of all the goals, domains and requirements using the Alloy model.

Effort spent
A complete table of all the hours spent by the team during the project.

2 | Overall description

2.1 Product perspective

The *Travlender* interacts with the system using an application on his/her smartphone. The user interface is designed for Android 7.1.1 (Nougat) or above. The application leans on a third-party *Transport Managing System* to handle all the payments related with public vehicles tickets and shared or non-shared vehicles books. It also leans on a third-party *Mapping Managing System* to handle the map, the path calculation algorithms and all the traffic or meteorological information.

2.2 Product functions

The application handles four typologies of event:

1. *Standard*: which provides calendar, reminder and street direction supports and cannot be overlapped with other activities.
2. *Lasting*: which provides calendar, reminder and street direction supports and can be overlapped with activities.
3. *Flexible*: which provides calendar, reminder and street direction supports and can be overlapped with activities as long as a minimum amount of reserved time fixed by the user exists.
4. *Transfer*: which provides calendar and reminder supports and cannot be overlapped with other activities. It is used for events that take place outside Milan.

Travlendar+ also provides in-app purchase for public transports tickets (metro, bus, trolleybus, tram, train) in Milan and bookings for shared (bike sharing, car sharing) and non-shared transports (taxi, limousine) services.

Travlendar+ takes account of different user preferences, like the opportunity to travel by owned means (*car*, *bike* or *foot*), the possibility to choose different algorithms to set the course (*quick*, *cheap* or *eco*) and offers the opportunity to reach the location choosing means that keeps the *Travlender* out of adverse weather conditions.

2.3 User characteristics

User
: A generic unregistered user, or a registered but unlogged user. At the application startup he/she can only tap on *Become a Travlender* or fullfill the login fields (in this case, he/she can alternatively tap on *Login with Facebook* or *Login with Google+*).

Travlender
: A registered and logged user. He/She has access to all the application functions.

2.4 Assumptions

2.4.1 Application assumptions

A.1 For the minimization of carbon footprint, all the public transports are considered like being zero-emission, since the user presence would not influence the vehicle emissions during its travel.

A.2 The cost of public trasports depends only on the tickets price and it is considered free if the user has already a pass.

A.3 The cost of a car travel is supposed to be the same for all cars and depends only on the distance.

A.4 For the same vehicle, the fastest way is even the most ecologic.

A.5 All the vehicles belonging to the same class have the same features (speed, pollution level, fuel cost, etc.).

2.4.2 Domain assumptions

D.1 The payment details are verified by a reliable third-party service.

D.2 The third-party payment service is always available.

D.3 The traffic and meteorological information comes from a third-party service always reliable.

D.4 The GPS position is always accurate as much as possible.

D.5 It is possible to keep track of the position of user means through a third-party service.

D.6 The cost for each car travel is given by the estimate of the kilometers per liter and euros per liter.

D.7 All the trams, busses, trolleybusses and metros are available for people with disabilities.

D.8 There are no unpredictable events that can cause any delay to the user (accidents, strikes, etc.).

D.9 Taxis and limousines follow the fastest, cheapest or most ecologic way, depending on the customer request.

D.10 Public transports are never full.

D.11 The owned bikes are considered to be handy and always available for the user.

2.4.3 Text assumptions

T.1 We consider a city as application's range of functionality, in particular the full support is given for the whole city of Milan.

T.2 The constraints about the time limitation of public means are chosen from the users and depends on their preferences, the availability of the travel means is considered separately.

3 | Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

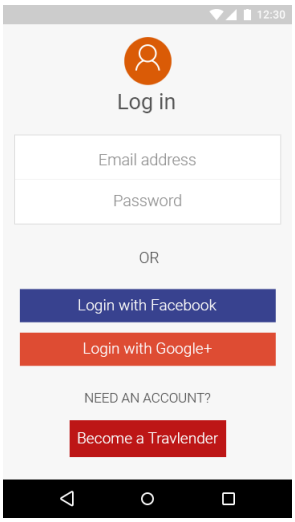


Figure 3.1: Login

The unregistered or unlogged user can log in to his/her account (if he/she has already got one) or create a new one. The application allows to log in with his/her own Facebook or Google+ account.

The first time the *Travlender* starts the application a pop-up appears, in which *Travlendar+* asks for the permission to use the user location. The user can tap on *allow* to give it, or *cancel* to refuse. In the second case, many application functions won't be accessible.

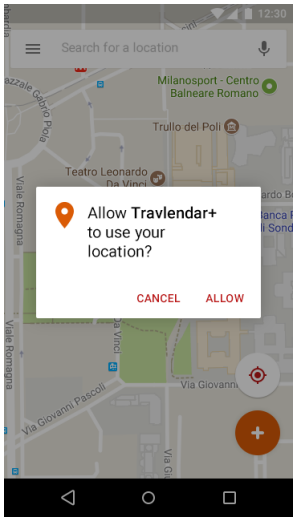


Figure 3.2: Alert

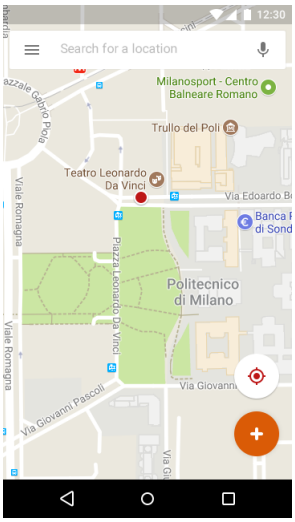


Figure 3.3: Map

If the *Travlender* decides to use the map view, he can see two buttons on the right. The first one centers the map on his/her position, the second one allows him/her to schedule a new event.

If the *Travlender* decides to use the calendar view instead, he can see only one button on the right that allows him/her to schedule a new event. The *Travlender* can go to the previous or next month swiping on the left or on the right, respectively.

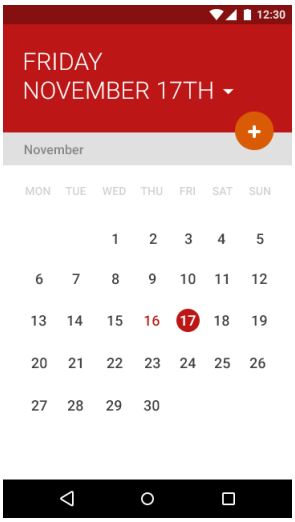


Figure 3.4: Calendar

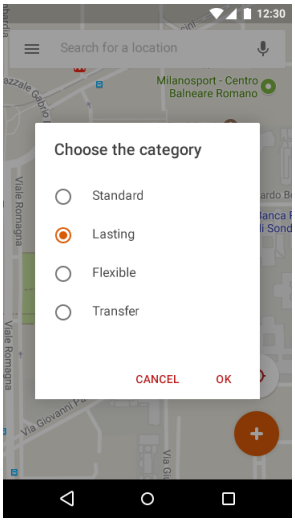


Figure 3.5: Choice of event category

When the *Travlender* decides to create a new event, he/she can tap the + button on the map and a pop-up appears, in which *Travlendar+* asks for the event category.

Now a new screen appears, in which the user can insert the location of the event, its name, a description, the starting and ending date and time and can modify the default settings, tapping the four buttons on the bottom. The first one permits to change the travel preference from *quick* to *cheap*, or *eco*. The second one permits to change the event type from *work* to *personal*. The last but one enables (or disables) the *not wet* travel. The last one enables (or disables) *notifications*.

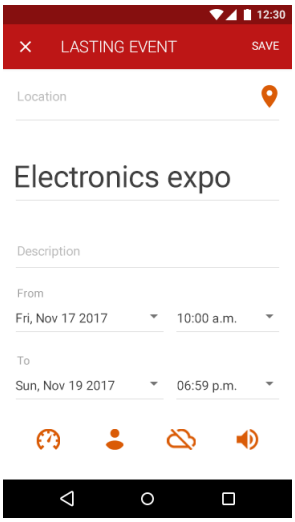
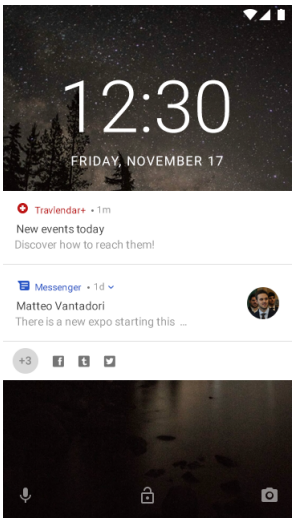


Figure 3.6: Creation of a new event



If there is at least a lasting event, a low-priority notification appears on the lock screen once a day.

Figure 3.7: Lock screen

The summary permits to modify the event information with the top-right button. The button on the right permits to change the location. Also, the *Travlender* can modify the default settings, tapping on the four buttons on the bottom. The first one is the *quick/cheap/eco* button. The second one is the *work/personal* button. The last but one is the *not wet* travel button. The last one is the *notifications* button. The bottom-right button is the *Go* button, tapping that the travel starts.

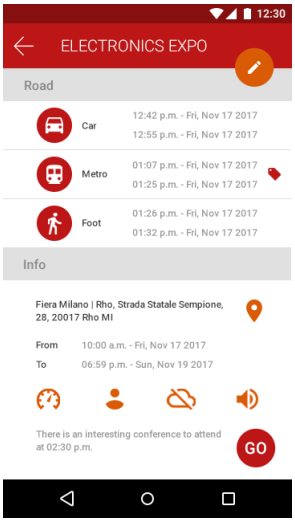


Figure 3.8: Summary of the event

3.1.2 Hardware interfaces

Travlendar+ doesn't need any hardware interface.

3.1.3 Software interfaces

Android 7.1.1 (or above)
Required operative system of user smartphone.

TMS
Third-party system which handles all the public, shared and non-shared vehicles purchase.

MMS
Third-party system which handles all the traffic and meteorological information and all the travel algorithms.

3.1.4 Communication interfaces

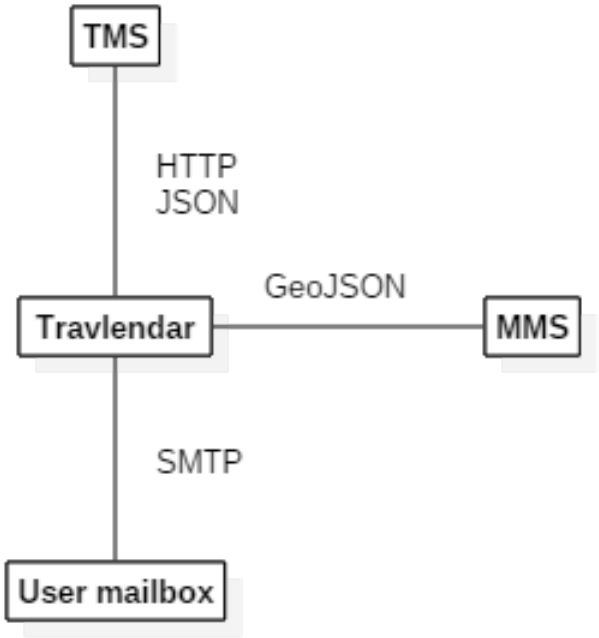


Figure 3.9: Interaction between the applications

TMS

JSON The system sends to the TMS the payment details in a JSON request body, the TMS sends back a JSON response body with the details.

HTTP Required for the *201 Created* status code.

MMS

GeoJSON The system sends to the MMS the user coordinates with the GeoJSON encoding format.

User mailbox

SMTP After the user registration, the system sends an email confirmation to the user mailbox.

3.2 Functional requirements

3.2.1 Functional requirements list

Section [R.1] treats the requirements related to the event scheduling:

R.1.1 The system must keep track of the user commitments.

Section [R.2] treats all the requirements related to the creation or modification of an event:

R.2.1 The system must verify that the *Travlender* is free during all the event time slot.

R.2.2 The system must verify that the *Travlender* is free during all the travel time slot.

R.2.3 The system must guarantee that the new event doesn't overlap the previous event time slot.

R.2.4 The system must avoid that the new event compromizes the reachability of the upcoming event.

R.2.5 The system must inform the *Travlender* when it's not possible to schedule the new event in the chosen time slot.

R.2.6 The system must insert the new event after having checked whether it's possible to do it.

R.2.7 The system must suggest possible solutions to arrange any overlap, either postponing the starting time of the upcoming event or anticipating the ending time of the previous one.

Section [R.3] treats all the requirements related to the public, shared and non-shared transports:

R.3.1 The system must locate all the bike sharing system in Milan on the map.

R.3.2 The system must locate all the car sharing system in Milan on the map.

R.3.3 The system must locate all the public transports stops in Milan on the map.

R.3.4 The system must unlock a shared car when required.

R.3.5 The system must unlock a shared bike when required.

R.3.6 The system must let the *Travlender* book a taxi via-app.

R.3.7 The system must let the *Travlender* book a limousine via-app.

Section [R.4] treats all the requirements related to the travel:

R.4.1 The system must control if the (eventually) required tickets are already owned by the *Travlender*.

R.4.2 The system must evaluate the best way to travel for the *Travlender*, according to his/her preferences, constraints and owned means.

R.4.3 The system must not suggest the *Travlender* to use the owned car and bike for a travel starting from a location where they aren't placed.

R.4.4 The system must allow the *Travlender* to purchase tickets via-app.

Section [R.5] treats all the requirements related to the special events:

R.5.1 The system must allow the *Travlender* to create an overlappable event.

R.5.2 The system must allow the *Travlender* to create an event with a specified reservation time slot.

Section [R.6] treats all the requirements related to the *Travlender* preferences:

R.6.1 The system must keep track of the *Travlender* public transports passes and tickets.

R.6.2 If asked, the system must notify the user if a scheduled event takes place on an adverse weather conditions day.

R.6.3 The system must allow a *Travlender* with disabilities to reach the event evaluating a way according to his/her needs.

R.6.4 The system must keep track of the means owned by the *Travlender*.

R.6.5 The system must keep track of the means selected by the *Travlender*.

R.6.6 The system must be able to evaluate the fastest way to reach the event, according to the *Travlender* preferences, constraints, owned means, tickets and passes.

R.6.7 The system must be able to evaluate the cheapest way to reach the event, according to the *Travlender* preferences, constraints, owned means, tickets and passes.

R.6.8 The system must be able to evaluate the most ecological way to reach the event, according to the *Travlender* preferences, constraints, owned means, tickets and passes.

3.2.2 Use case diagrams

User

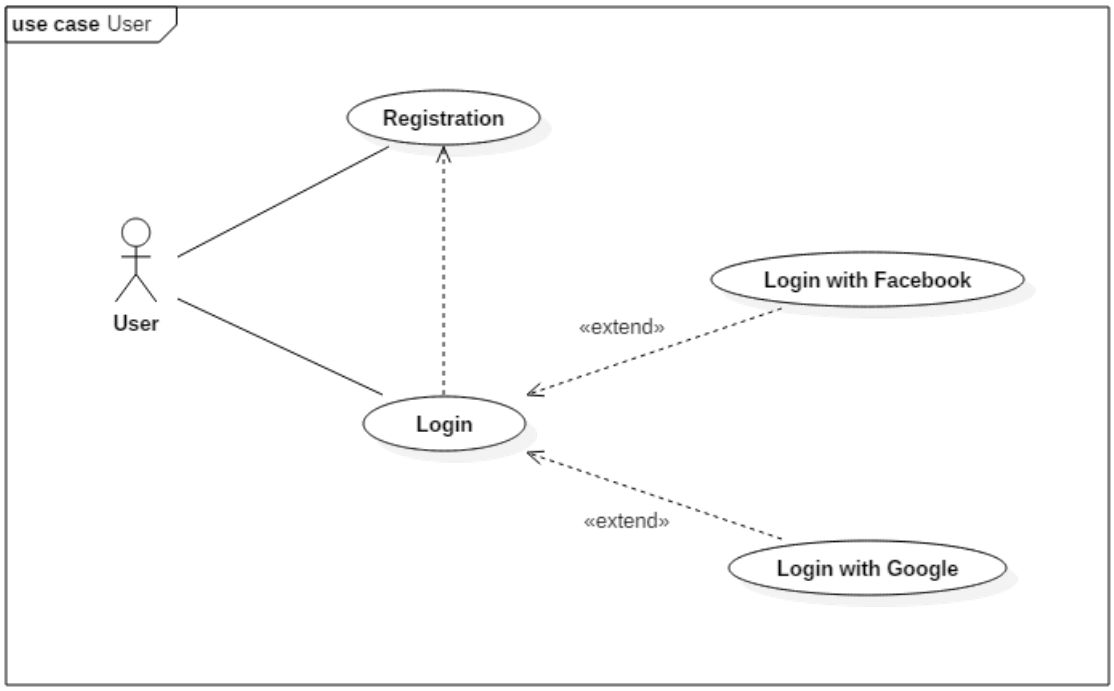


Figure 3.10: Use case diagram of the user

Travlender

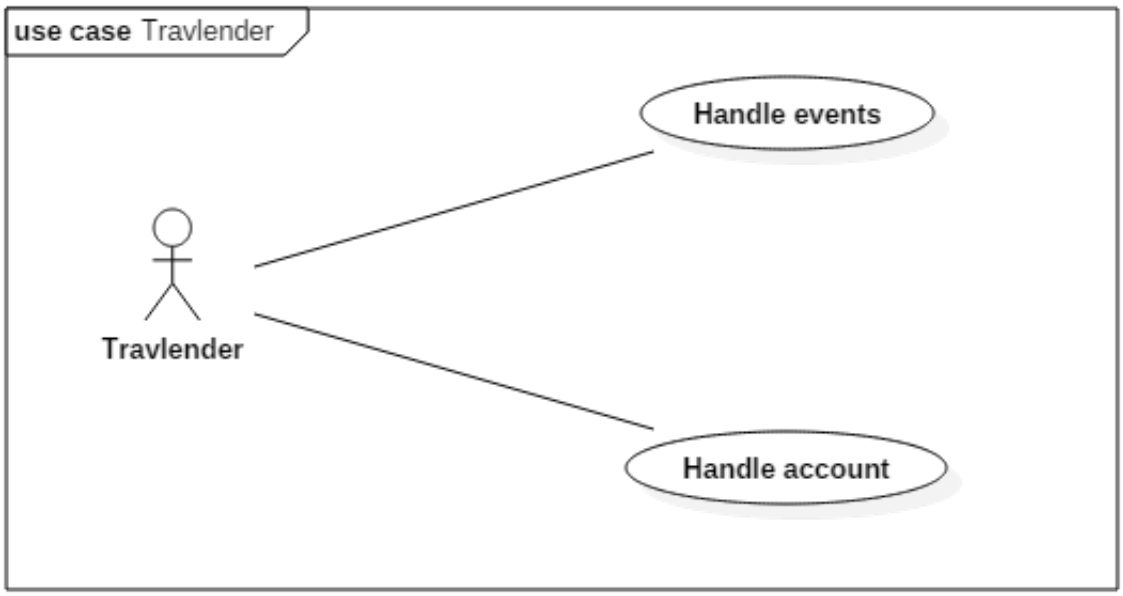


Figure 3.11: The *Travlender* actions are divided in two main categories

Handle events

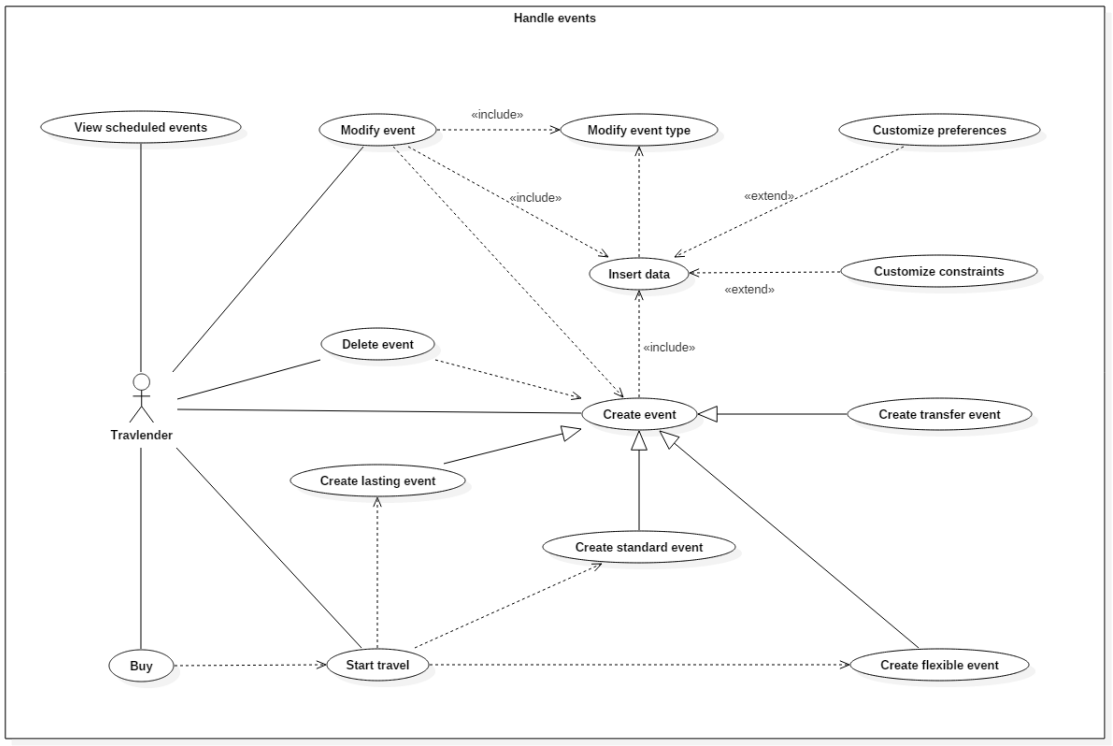


Figure 3.12: The *handle events* actions category

Handle account

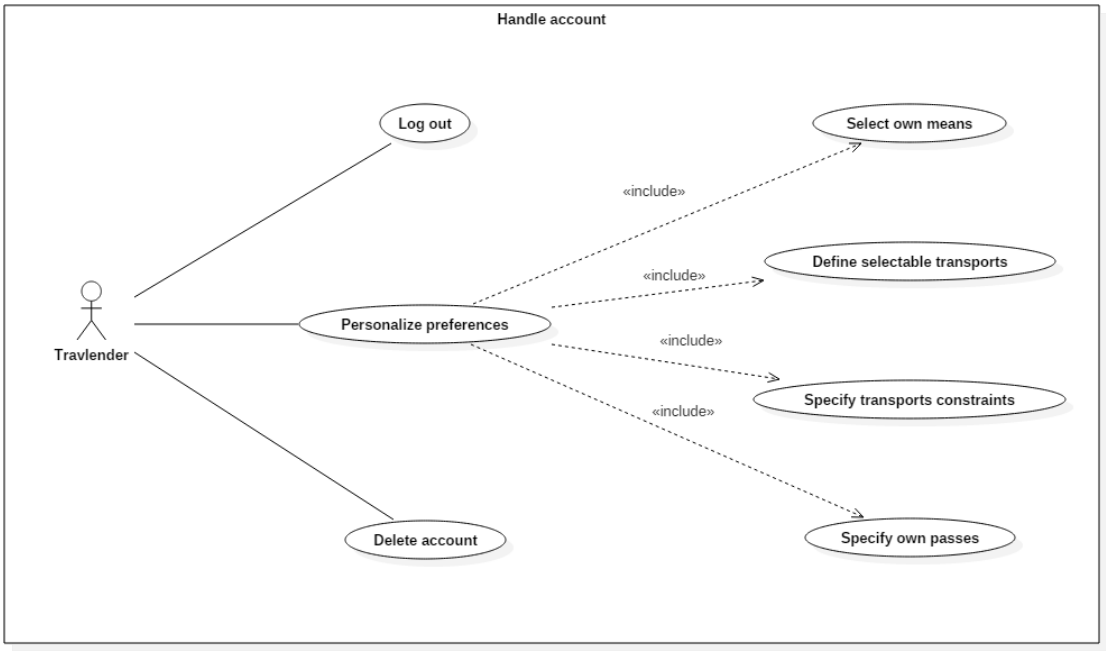


Figure 3.13: The *handle account* actions category

3.2.3 Use case tables

User registration

Name	Registration
Actors	User
Goals	-
Input conditions	The user must have downloaded the application and opened it on the login screen
Events flow	1. The user taps *Become a Travlender* 2. The user fullfills the registration form and taps *Send* 3. The system verifies the data 4. The system shows a confirm message on the screen
Output conditions	The user is registered
Exceptions	The user makes a mistake in the registration form: 4.a. The system shows an error message on the screen Then the flow restarts from point 2

User login

Name	Login
Actors	User
Goals	-
Input conditions	The user must be registered and have opened the appliation on the login screen
Events flow	1. The user fills out *Email address* and *Password* fields and taps *Enter* 2. The system verifies the account 3. The system shows a confirm message on the screen
Output conditions	The user is logged
Exceptions	The user inserts an invalid email address or password: 3.a. The system shows an error message on the screen Then the flow restarts from point 1

User login with Facebook

Name	Login with Facebook
Actors	User
Goals	-
Input conditions	The user must be registered and have opened the appliation on the login screen
Events flow	1. The user taps *Login with Facebook* 2. The system verifies the account 3. The system shows a confirm message on the screen
Output conditions	The user is logged
Exceptions	The user hasn't go an associated Facebook account: 3.a. The system shows an error message on the screen Then the flow restarts from point 1

User login with Google+

Name	Login with Google+
Actors	User
Goals	-
Input conditions	The user must be registered and have opened the appliation on the login screen
Events flow	1. The user taps *Login with Google+* 2. The system verifies the account 3. The system shows a confirm message on the screen
Output conditions	The user is logged
Exceptions	The user hasn't go an associated Google+ account: 3.a. The system shows an error message on the screen Then the flow restarts from point 1

View scheduled events

Name	View scheduled events
Actors	Travlender
Goals	G.1.11
Input conditions	The user must be logged
Events flow	1. The Travlender swipes the screen on the left and a menu appears 2. The *Travlender* taps Scheduled events
Output conditions	The *Travlender* can view the scheduled events
Exceptions	There are no exception cases

Create a new standard event

Name	Create standard event
Actors	Travlender
Goals	G.1.1 G.1.9
Input conditions	The user must be logged
Events flow	1. The *Travlender* taps the + button 2. The *Travlender* taps the *Standard event* category 3. The *Travlender* inserts the name 4. The *Travlender* inserts starting time and ending time 5. The *Travlender* inserts a description (optional) 6. The *Travlender* modifies the constraints (optional) 7. The *Travlender* adds a location 8. The *Travlender* taps *Save*
Output conditions	A new standard event is created
Exceptions	The *Travlender* inserts a time slot overlapped with another standard event: 5.a. The system shows an error message on the screen Then the flow restarts from point 4
	The *Travlender* inserts a non-existent location: 8.b. The system shows an error message on the screen Then the flow restarts from point 7

Create a new lasting event

Name	Create lasting event
Actors	Travlender
Goals	G.1.1 G.1.9
Input conditions	The user must be logged
Events flow	1. The *Travlender* taps the + button 2. The *Travlender* taps the *Lasting event* category 3. The *Travlender* inserts the name 4. The *Travlender* inserts starting time and ending time 5. The *Travlender* inserts a description (optional) 6. The *Travlender* modifies the constraints (optional) 7. The *Travlender* adds a location 8. The *Travlender* taps *Save*
Output conditions	A new lasting event is created
Exceptions	The *Travlender* inserts a non-existent location: 8.a The system shows an error message on the screen Then the flow restarts from point 7

Create a new flexible event

Name	Create flexible event
Actors	Travlender
Goals	G.1.1 G.1.9
Input conditions	The user must be logged
Events flow	1. The *Travlender* taps the + button 2. The *Travlender* taps the *Flexible event* category 3. The *Travlender* inserts the name 4. The *Travlender* inserts starting time and ending time 5. The *Travlender* inserts a minimum time slot 6. The *Travlender* inserts a description (optional) 7. The *Travlender* modifies the constraints (optional) 8. The *Travlender* adds a location 9. The *Travlender* taps *Save*
Output conditions	A new flexible event is created
Exceptions	The *Travlender* inserts a time slot bigger than the event duration: 6.a The system shows an error message on the screen Then the flow restarts from point 5
	The *Travlender* inserts a non-existent location: 9.b The system shows an error message on the screen Then the flow restarts from point 8

Create a new transfer event

Name	Create transfer event
Actors	Travlender
Goals	G.1.1
Input conditions	The user must be logged
Events flow	1. The *Travlender* taps the + button 2. The *Travlender* taps the *Transfer event* category 3. The *Travlender* inserts the name 4. The *Travlender* inserts starting time and ending time 5. The *Travlender* inserts a description (optional) 6. The *Travlender* adds a location 7. The *Travlender* taps Save
Output conditions	A new transfer event is created
Exceptions	The *Travlender* inserts a non-existent location: 7.a The system shows an error message on the screen Then the flow restarts from point 6

Delete an existent event

Name	Delete event
Actors	Travlender
Goals	G.1.10
Input conditions	The user must be logged and must exist at least an event
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Scheduled events* 3. The *Travlender* taps the event he wants to delete 4. The *Travlender* taps *Delete*
Output conditions	A new transfer event is created
Exceptions	There are no exception cases

Modify an existent event

Name	Modify event
Actors	Travlender
Goals	G.1.2 G.1.3 G.1.4 G.1.5 G.1.6 G.1.7 G.1.8
Input conditions	The user must be logged and at least an event must exist
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Scheduled events* 3. The *Travlender* taps the event he wants to modify 4. The *Travlender* modifies the desired fields 5. The *Travlender* taps *Save*
Output conditions	A new transfer event is created
Exceptions	The *Travlender* inserts a non-valid field: 5.a The system shows an error message on the screen Then the flow restarts from point 4

Start travel of an existing event

Name	Start travel
Actors	Travlender
Goals	G.1 G.2 G.3 G.4 G.5 G.6 G.7 G.8
Input conditions	The user must be logged and at least an event must exist
Events flow	1. The system warns the *Travlender* that an event is about to start 2. The *Travlender* taps *Go*
Output conditions	The event starts
Exceptions	The *Travlender* taps Go when he can no longer reach the event before the start, but he can still reach it before the end: 2.a. The system warns the *Travlender* he/she is late Then the flow restarts from point 2
	The *Travlender* taps *Go* when he can no longer reach the event before the end: 2.b The system warns the *Travlender* he/she can no longer reach the event before the end Then the flow ends

Buy a ticket or book a mean

Name	Buy
Actors	Travlender
Goals	G.4.1 G.4.2 G.5.1 G.5.2 G.5.3 G.5.4 G.5.5 G.6.1 G.6.2
Input conditions	The user must be logged and the event must be started
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Scheduled events* 3. The *Travlender* taps the ongoing event 4. The *Travlender* taps the public mean he/she wants to buy a ticket or the mean he/she wants to book 5. The *Travlender* taps *Buy/Book*
Output conditions	The *Travlender* buys the ticket or books the mean
Exceptions	There are no exception cases

Select own means

Name	Select own means
Actors	Travlender
Goals	-
Input conditions	The user must be logged
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Personalize* 3. The *Travlender* taps *Own means* 4. The *Travlender* selects his/her own means from a list 5. The *Travlender* taps *Save*
Output conditions	*Travlendar+* knows the user own means
Exceptions	There are no exception cases

Define selectable transports

Name	Define selectable transports
Actors	Travlender
Goals	-
Input conditions	The user must be logged
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Personalize* 3. The *Travlender* taps *Selectables* 4. The *Travlender* selects the means the app has to consider in his/her travels 5. The *Travlender* taps *Save*
Output conditions	*Travlendar+* knows the means to consider in travels
Exceptions	There are no exception cases

Specify transports constraints

Name	Specify transports constraints
Actors	Travlender
Goals	-
Input conditions	The user must be logged
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Personalize* 3. The *Travlender* taps *Constraints* 4. The *Travlender* taps the vehicle desired from a list 5. The *Travlender* inserts the constraints in the field 6. The *Travlender* taps *Save*
Output conditions	*Travlendar+* knows the means constraints to consider in travels
Exceptions	There are no exception cases

Specify own public vehicle passes

Name	Specify own passes
Actors	Travlender
Goals	-
Input conditions	The user must be logged
Events flow	1. The *Travlender* swipes the screen on the left and a menu appears 2. The *Travlender* taps *Personalize* 3. The *Travlender* taps *Passes* 4. The *Travlender* select the passess he has from a list 5. The *Travlender* taps *Save*
Output conditions	*Travlendar+* knows which passes the user has
Exceptions	There are no exception cases

3.2.4 Class diagram

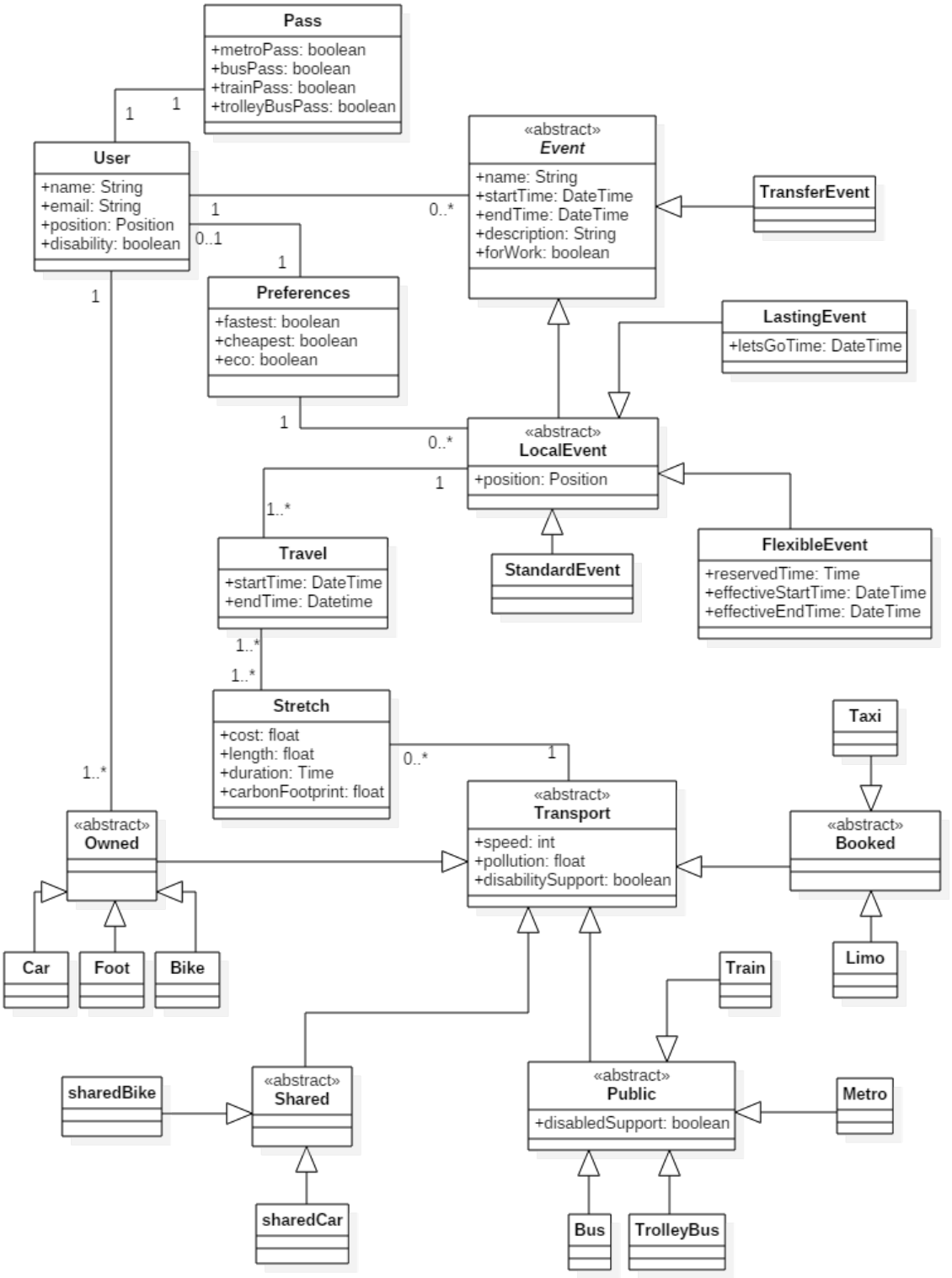


Figure 3.14: Class diagram

3.2.5 Sequence diagrams

User registration

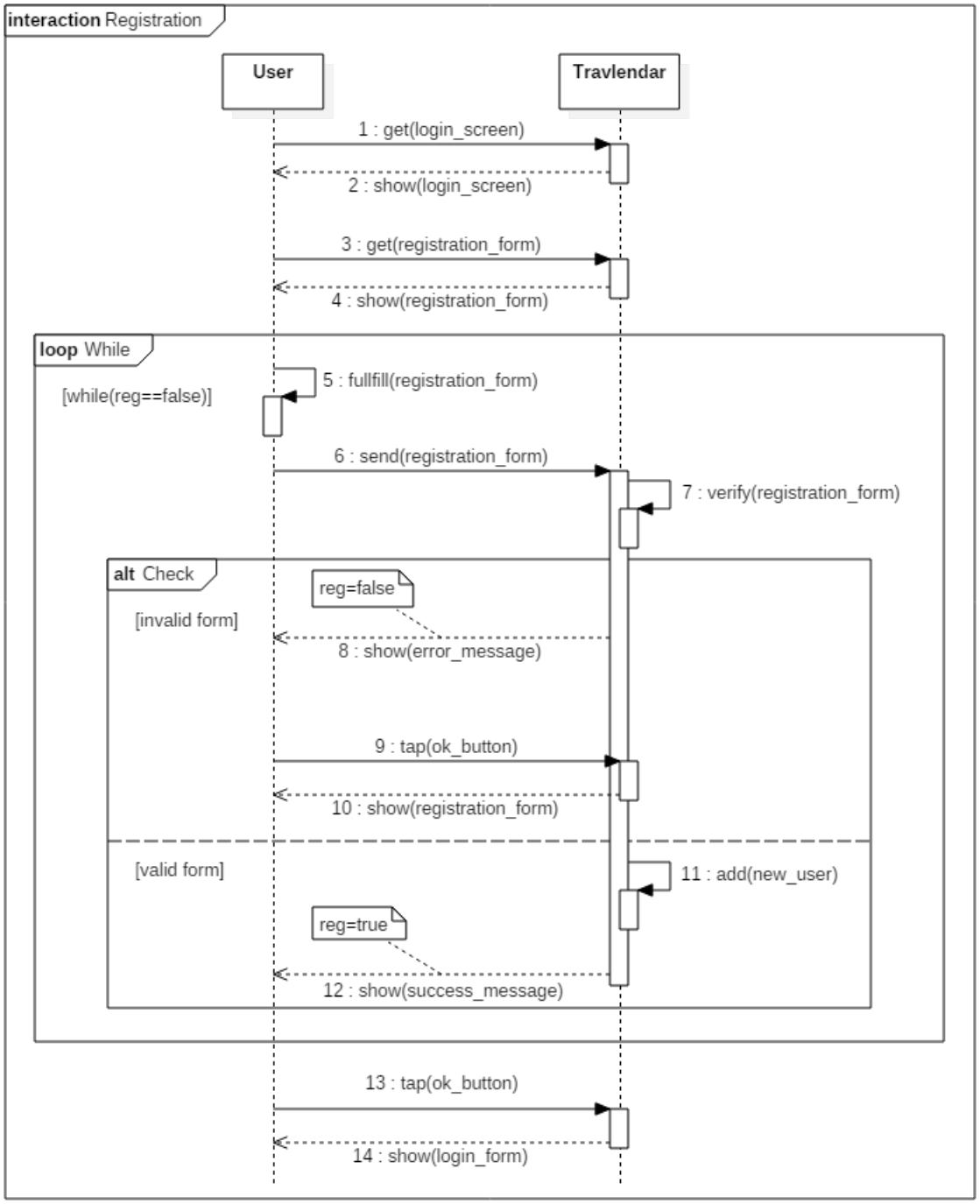


Figure 3.15: User registration diagram

User login

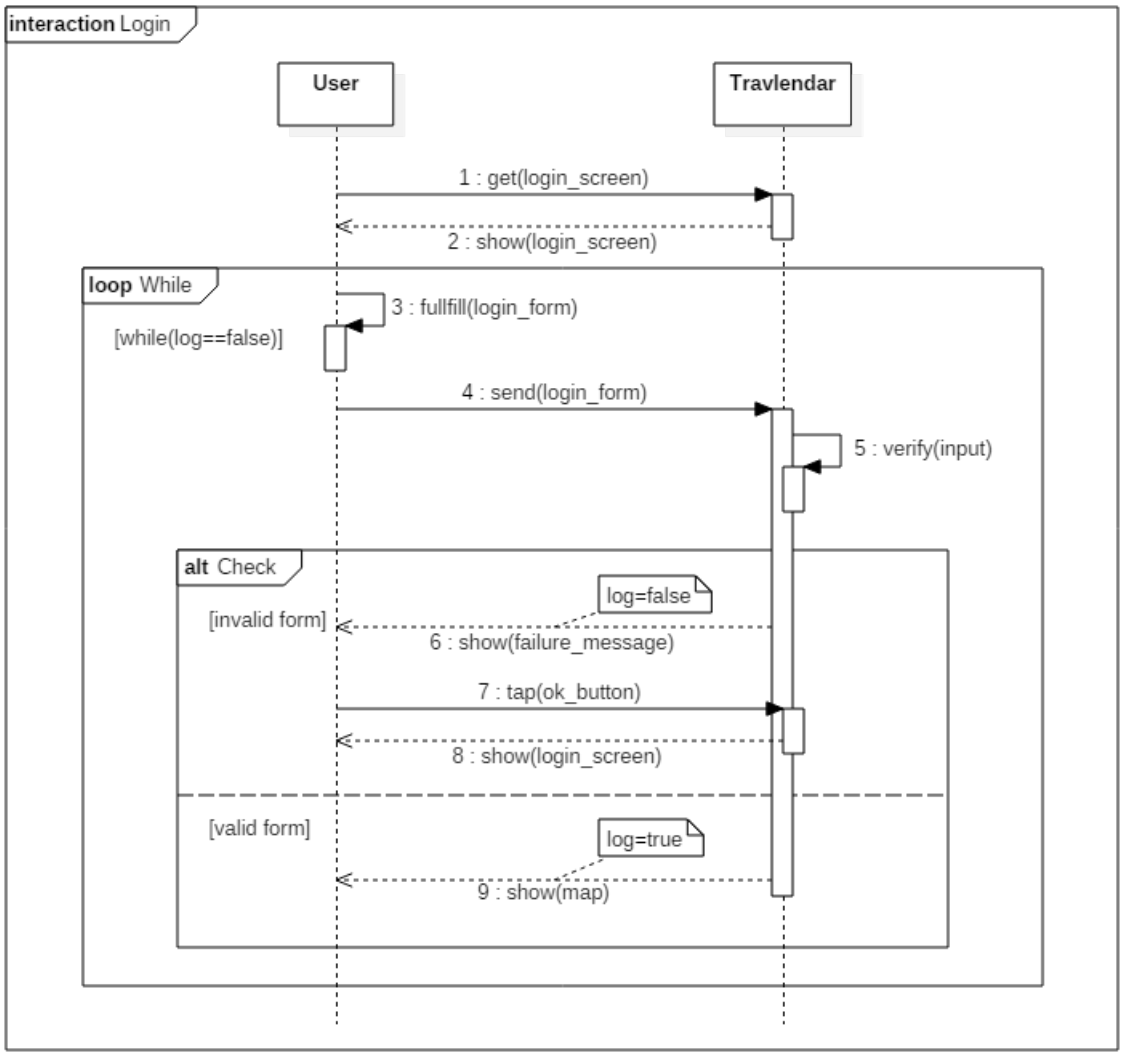


Figure 3.16: User login diagram

Event creation and edit

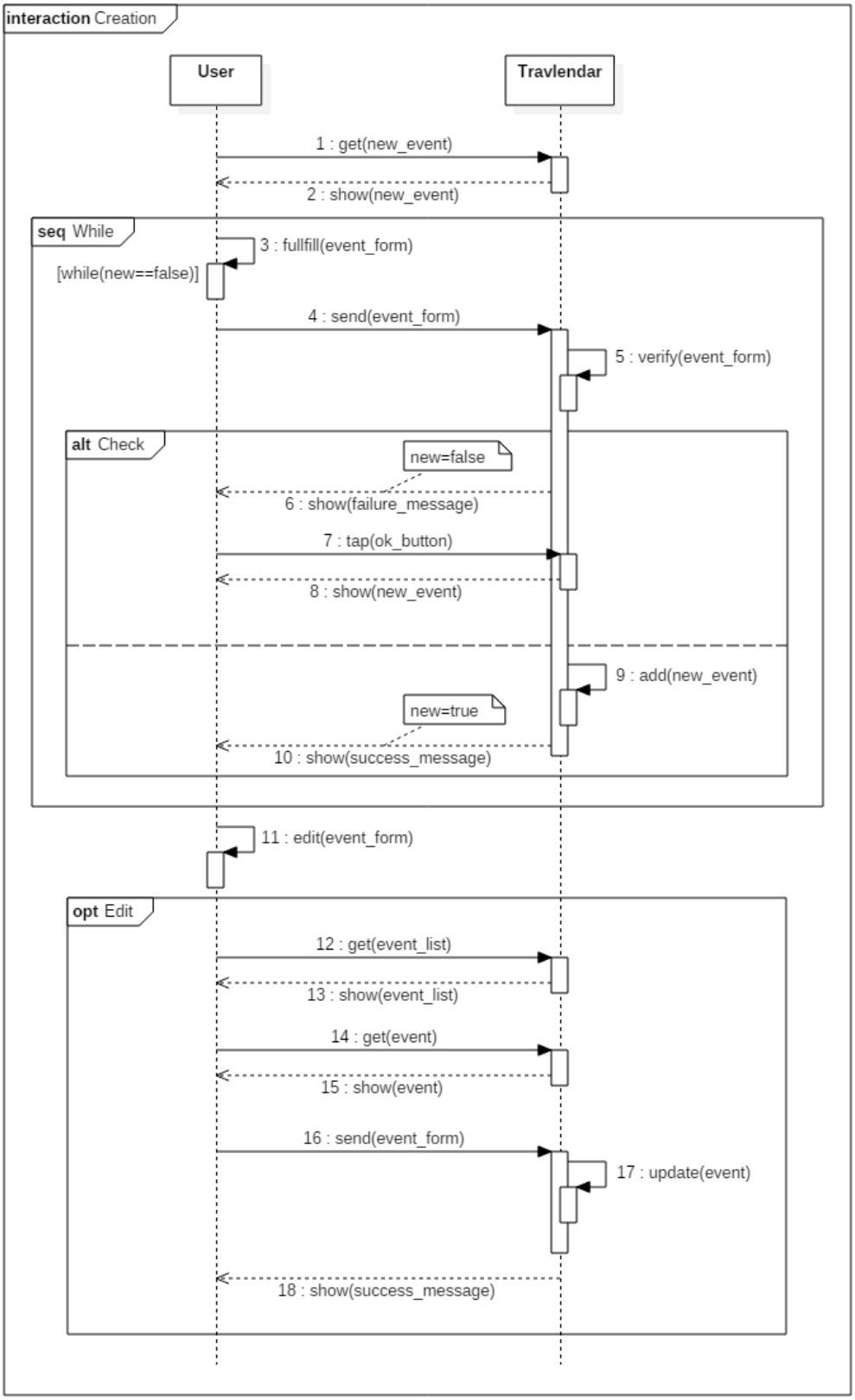


Figure 3.17: Event creation and edit diagram

Ticket purchase

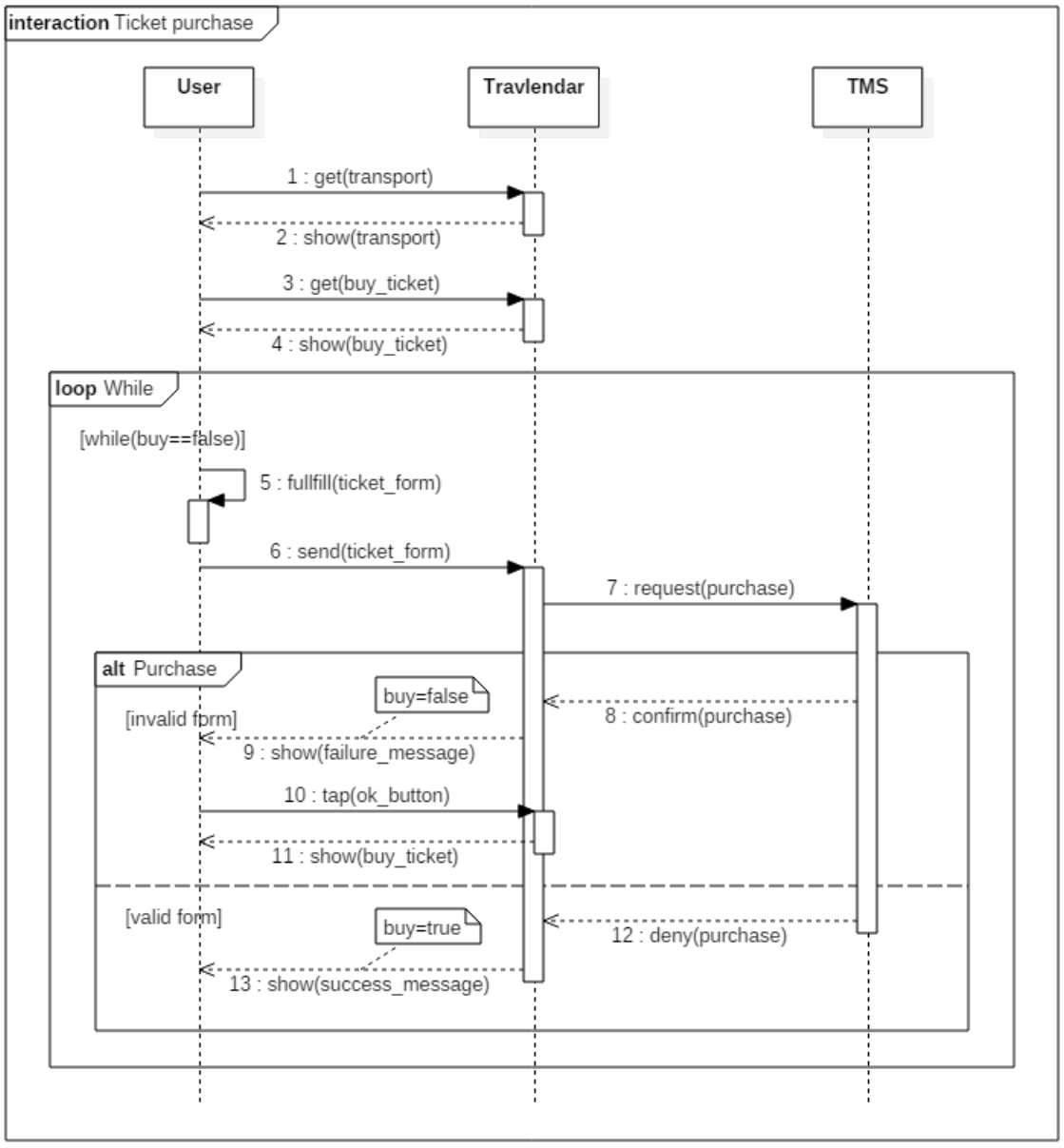


Figure 3.18: Ticket purchase diagram

3.3 Design constraints

3.3.1 Hardware limitations

- Android 7.1.1 (or above) is recommended.
- GPS has to be always aviable.
- The system requires at least a 3G wireless technology.
- A minimum space for the application download is required.

3.4 Software system attributes

3.4.1 Reliability

The MMS has to offer a precise GPS service.

3.4.2 Availability

Travlendar+ is designed to be a 24/7 service.

3.4.3 Security

- *Travlendar+* must allow the unregistered user to sign up, checking the correctness of his/her registration form.
- *Travlendar+* must allow the registered user to sign in, checking his/her credentials.
- *Travlendar+* must allow the registered and logged user to sign out.
- *Travlendar+* must allow the registered and logged user to delete his/her account, checking his/her credentials.

3.4.4 Maintainability

Short maintenance periods are allowed.

4 | Scenarios

4.1 Scenario 1

Registration and calendar function
Tom is a student who recently moved to Milan for his studies. He is a messy boy and always had his events scattered on post-its all over the house. This condition often made him miss appointments. Carlo, his roommate, tired of the disorder, presented to him *Travlendar+*, a new born application suitable for situations as Tom's.
Tom downloaded the application and after a quick registration via Facebook he transferred all the events he had hanging around in his *Travlendar+* app. He was alerted when events where set in an overlapping moment and always notified in time for his events.
In few days he gladly noticed how having all his events always under control brought an improvement in his daily routine organization. Even Carlo was happy, he could finally throw away all the post-it notes.

4.2 Scenario 2

Flexible event
Lucy is a mother who enjoys cooking but she lately spent much time complaining about her old pots. Since her birthday was about to come, her son Mark decided to buy her a new set of pots. He recently started working, and with his other daily commitments he knew that he does not have much free time. He realized that his only chance might be on Wednesday, when there is the open market in the next street.
Therefore he opened *Travlendar+* and tried to fit the new commitment in his day schedule. As he knew that the market started at 8.00 a.m. and lased at 5.00 p.m. he scheduled a new flexible event in the mentioned hours and set an occupation time of thirty minutes, time he needed to find the best pots for his mom. On Wednesday the idea of the present was already out of his mind when, after a busy morning, *Travlendar+* notified him that he had the free time to spend at the market. Mark went to buy the pots and after giving them to his mother Lucy, she was so happy that she decided to inaugurate them with a cake.

4.3 Scenario 3

Transfer event, preferences edit for single travel
George had been invited by his friend Lucas to have lunch together in a restaurant in Monza.
He opened *Travlendar+* and exploring the map he found the restaurant and added his commitment from the map screen. The new event was automatically scheduled as a transfer event because Travlendar+ recognized the location as outside the city of Milan. George was therefore warned that, since it is a transfer, *Travlendar+* would not give full support for the travel: it wouldn't be able to suggest any route for the destination and during all the time spent outside the city it wouldn't manage to compute any travel time for future appointments.
George was a bit worried, since he was using the application on which he has always relied on, but fortunately he is a long-standing client of *Travlendar+* so he knew all the various features of the application, so before leaving he checked his next appointment: a dinner, that same day at

his grandma's house in Milan. He estimated that by car the travel from the restaurant to the grandma's house would take an hour long. Then he just opened the event set at grandma's and modified the event's preferences, setting an one hour reminder.
So he could leave the city without worries, aware that the evening *Travlendar+* would notify him in time.

4.4 Scenario 4

Lasting event, *Go* button, event editing
Every year in Milan there is an electronics fair where you can find all kind of electronics products and services. Mike is passionate about electronics so he would happily attend some stands if he finds the occasion and this year may be the right one since the exposition has the special duration of one week.
Mike scheduled the occasion as a lasting event indicating the usual information where he specified the days of beginning and ending of the fair. The starting day of the fair, a notification popped up on Mike's phone to remind him of the event.
Mike was excited because he discovered that it was supposed to be a presentation about virtual reality he wanted to attend, so as soon as he finished his work, he decided to move to the fair. He opened the event and tapped *Go*. Unfortunately the application warned him that it was not possible to start the trip because the time needed to get there wouldn't allow him to arrive on time to the next event.
Indeed Mike was forgetting that the same night he would have an appointment at the theatre with his friend Carl. After a moment, he decided to give the priority to the fair so he calls his friend in order to set a new appointment. They confirmed the location of theatre but had to change the previously decided place, because the former one didn't have interesting shows during the day of the new appointment.
Mike modified the theatre event, changed time and place of the meeting and added a description as a reminder not to miss the appointment. After the confirmation of the edit, Mike could start his journey to the electronic fair.
On the next day, in the morning Mike saw a low-priority notification that informed him there were ongoing events: it was still the electronic fair. That time he just ignored it because he couldn't disappoint his friend.

4.5 Scenario 5

Multi-transport function, preferences edit, ticket purchase, means constraints
Carl is an environmental lawyer, and had to take part to an important conference about the sustainable development. During his speech he had an excursus about how the *Travlendar+* application helped him get to the conference location that same night. He talked about how he valued the possibility that the application gave him that morning in letting him choose the most ecological route.
He took a one hour long travel, as suggested by *Travlendar+*. As he confirmed to follow the suggested route Carl was notified that a metro ticket was needed for the travel. He was unlucky because if the transport was a train, tickets wouldn't have been a problem since he added in his *Travlendar+* preferences a train subscription, valid for the whole year.
There weren't big deals because *Travlendar+* offers the opportunity to buy the tickets in-app and so he did it in just a few seconds thanks to some valid payment services used by the application. So he received an e-mail which notified him that the operation has been successfully done.
He then started the journey on his bike, and after having pedaled for ten minutes he got to the nearest tram stop. He got to the terminus of the tram after thirty minutes, and while returning on his bike for the last stretch, as the app suggested it him, he saw another tram, leaving that same stop with the conference venue displayed as destination. Despite that he was not upset, because he does not like to take trams at that time of the night because of some bad experiences he had. Only later he remembered that it was him who added the time constrain on transport.

4.6 Scenario 6

Disability support, non-shared transports booking
Louis is a biker who, due to an accident, was forced to use the wheelchair for a couple of months. During this period he found hard being independent since he could not move on his own.
Luckily, he discovered *Travlendar+* and, thanks to its services, he set the disability option, in order to use only transports that could support his problem. One day he had an appointment with his new girlfriend and used *Travlendar+* to reach her.
Not only did *Travlendar+* suggest him a limousine which had the disability support but also offered him the chance to book it via-app. Louis confirmed, sure to impress his girlfriend.

5 | Formal analysis using alloy

5.1 Source code

```
open util/boolean

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

one sig User{
        event: set Event,
        pass: lone Pass,
        preference: one Preference,
        position: one Position,
        disability: lone Bool,
        maxFootVal, maxCostVal: lone Int,
        safePublicTimeRide: lone Bool,
        rainPrevention: lone Bool
}{
        maxFootVal ≥ 0
        maxCostVal ≥ 0
}

one sig Weather{
        rainyHours: set Rain
}

sig Rain{
        startingTime: Int,
        endingTime: Int
}{
        startingTime ≥ 0
        endingTime > startingTime
}

sig Preference{
        fastest, cheapest, eco: lone Bool
}{
        //one and only one preference will be chosen
        fastest = True implies (no cheapest and no eco)
        eco = True implies (no cheapest and no fastest)
        cheapest = True implies (no fastest and no eco)

        fastest = True or cheapest = True or eco = True
}

sig Pass{
        metroPass, busPass, trainPass, trolleyBusPass: lone Bool
}{
        //when the passes exist they are always valid (true)
        metroPass = True and  busPass = True and trainPass = True and
            ↪ trolleyBusPass = True
}

abstract sig Event {
        startTime, endTime: one Int
}{
        startTime > 0
```

```
        endTime > 0
        startTime < endTime
}

sig TransferEvent extends Event{
}

abstract sig LocalEvent extends Event{
        travel: some Travel,
        chosenTravel: one Travel,
        location: one Position,
        preference: lone Preference,
        travelTime: Int
}{
        all t: travel | t.lastStretch.endPosition = location
        chosenTravel in travel
}

sig StandardEvent extends LocalEvent{
}

sig FlexibleEvent extends LocalEvent{
        reservedTime: Int,
        effectiveStartTime: Int,
        effectiveEndTime: Int
}{
        reservedTime = sub[effectiveEndTime, effectiveStartTime]
        reservedTime > 0
        effectiveStartTime > 0
        effectiveEndTime > 0
        effectiveStartTime ≥ startTime
        effectiveEndTime ≤ endTime
}

sig LastingEvent extends LocalEvent{
        letsGoButton: set LetsGo
}{
        all g: letsGoButton | g.referingEvent = this
}

sig LetsGo{
        letsGoTime: Int,
        chosenTravel: one Travel,
        referingEvent: one LastingEvent
}{
//the Go button can be tapped when the suggested travel let the user arrive during
    ↪ the event interval
        letsGoTime ≥ sub[referingEvent.startTime, chosenTravel.totalDuration]
        letsGoTime < sub[referingEvent.endTime, chosenTravel.totalDuration]

        chosenTravel.lastStretch.endPosition = referingEvent.location
        letsGoTime > 0
}

sig Travel{
        stretch: some Stretch,
        lastStretch: one Stretch,
        totalCost: Int,
        totalCFP: Int,
        totalDuration: Int,
        transport: some Transport,
        foot: set Foot,
        footLength: Int
}{
        totalCost = computeCost[stretch]
        totalCFP = computeTotalCFP[stretch]
        totalDuration = computeDuration[stretch]
        totalCost ≥ 0
        totalCFP ≥ 0
        totalDuration > 0
        footLength ≥ 0
```

```
        footLength = computeFootLength[foot]

        lastStretch in stretch
        all s: stretch | s.transport in transport
        no t:transport | (no s: stretch | s.transport = t)
        all disj s1,s2: stretch | s1.transport ≠ s2.transport
        no disj x1 ,x2: stretch | x1.startPosition = x2.startPosition
        no disj x1,x2: stretch | x1.endPosition = x2.endPosition
        one x: stretch | x.startPosition = User.position

        //if a stretch is not the first one, than there is another stretch which
            ↪ ends where and when the first one starts
        all x1: stretch |  x1.startPosition ≠ User.position implies (one x2:
            ↪ stretch | (x1.startPosition = x2.endPosition and x1.startTime = x2.
            ↪ endTime))
        //if a stretch is not the last one, than there is another stretch which
            ↪ starts where and when the first one ends
        all x1: stretch | x1 ≠ lastStretch implies (one x2: stretch | (x1.
            ↪ endPosition = x2.startPosition and x1.endTime = x2.startTime))
}

sig Stretch{
        cost, length, duration, CFP: Int,
        startPosition, endPosition: Position,
        transport: one Transport,
        startTime, endTime: Int
}{
        add[startTime, duration] = endTime
        startTime > 0
        endTime > 0
        duration > 0
        duration < 5
        cost ≥ 0
        CFP ≥ 0
        length > 0
        length < 5
        startPosition ≠ endPosition
        CFP = computeCFP[transport.pollution, length]

        //for the owned cars and booked vehicles the cost depends on the distance (
            ↪ fuel, etc.)
        transport in Car or transport in Booked implies cost = mul[length,
            ↪ transport.cost]
        //for the shared vehicles the cost depends on the time of use
        transport in Shared  implies cost = mul[duration, transport.cost]
        //for the public vehicles the cost depends on the vehicle itself
        transport in Public implies cost = transport.cost

        length = transport.meanDistance
}

abstract sig Transport{
        pollution, cost: Int,
        disabilitySupport: lone Bool,
        meanDistance: Int
}{
        pollution ≥ 0
        pollution < 5
        cost < 5
}

abstract sig Booked extends Transport{}
{
        pollution > 0
        cost > 0
}

sig Taxi extends Booked{}

sig Limo extends Booked{}
```

```
abstract sig Shared extends Transport{}
{
        pollution = 0
        cost > 0
        no disabilitySupport
}

sig SharedBike extends Shared{}

sig SharedCar extends Shared{}

abstract sig Owned extends Transport {}

sig Bike extends Owned{}
{
        pollution = 0
        cost = 0
        no disabilitySupport
}

sig Car extends Owned{}
{
        pollution > 0
        cost > 0
}

sig Foot extends Owned{}

{
        pollution = 0
        cost = 0
        no disabilitySupport
}

abstract sig Public extends Transport {}
{
        disabilitySupport = True
        pollution = 0
        cost ≥ 0
}

sig Metro extends Public{}

sig Bus extends Public{}

sig Train extends Public{}

sig TrolleyBus extends Public{}

sig Position{}

/*************************************FACT*****************************************/

//each local event belongs to one of the categories
fact {
        Event = LocalEvent + TransferEvent
        LocalEvent = StandardEvent + FlexibleEvent + LastingEvent
        Transport = Booked + Shared + Owned + Public
    Booked = Taxi + Limo
    Owned = Foot + Bike + Car
    Public = Bus + Train + Metro + TrolleyBus
    Shared = SharedCar + SharedBike
}

//we represent only the used elements
fact{
        all s: Stretch | s in Travel.stretch
        all p: Pass | p in User.pass
        all t: Travel | t in StandardEvent.travel or t in FlexibleEvent.travel or t
            ↪  in LetsGo.chosenTravel
        all e: Event | e in User.event
```

```
        all t: Transport | t in Stretch.transport
        all p: Position | (p in Stretch.startPosition or p in Stretch.endPosition)
        all b: Bool | ((b in Pass.metroPass or b in Pass.busPass or b in Pass.
            ↪ trainPass or b in Pass.trolleyBusPass or b in User.disability or b
            ↪ in Transport.disabilitySupport or
                    b in Preference.eco or b in Preference.fastest or b in
                        ↪ Preference.cheapest or b in User.safePublicTimeRide
                        ↪ or b in User.rainPrevention))
        all p: Preference | p in User.preference or p in LocalEvent.preference
        all r: Rain | r in Weather.rainyHours
        all l: LetsGo | l in LastingEvent.letsGoButton
}

//all the transports of the same type have the same pollution level
fact {
        all t1, t2 : Taxi | t1.pollution = t2.pollution
        all l1, l2 : Limo | l1.pollution = l2.pollution
        all c1, c2 : Car | c1.pollution = c2.pollution
}

//the user pass implies no cost for the relative public transport and the user
    ↪ without the pass implies a positive cost for the relative public transport
fact{
        User.pass.metroPass = True iff Metro.cost = 0
        User.pass.trainPass = True iff Train.cost = 0
        User.pass.trolleyBusPass = True iff TrolleyBus.cost = 0
        User.pass.busPass = True iff Bus.cost = 0
}

//the reserved time in a flexible event is always less than the event time slot
fact{
        all f: FlexibleEvent | f.reservedTime < sub[f.endTime, f.startTime]
}

//when a preference for the event is set, the best way to travel is found according
    ↪  to that preference
fact{
        all e: LocalEvent | e.preference.eco = True implies
                (all t: e.travel | t.totalCFP ≥ e.chosenTravel.totalCFP)
}

fact{
        all e: LocalEvent | e.preference.cheapest = True implies
                (all t: Travel | t in e.travel implies t.totalCost ≥ e.chosenTravel
                    ↪ .totalCost)
}

fact{
        all e: LocalEvent | e.preference.fastest = True implies
                (all t: Travel | t in e.travel implies t.totalDuration ≥ e.
                    ↪ chosenTravel.totalDuration)
}

//when no preference for a single event is set, the best way to travel is found
    ↪ according to the user default preference
fact{
        all e: LocalEvent | (allFalse[e] = True and User.preference.eco = True)
            ↪ implies
                (all t: Travel | t in e.travel implies t.totalCFP ≥ e.chosenTravel.
                    ↪ totalCFP)
}

fact{
        all e: LocalEvent | (allFalse[e] = True and User.preference.fastest = True)
            ↪  implies
                (all t: Travel | t in e.travel implies t.totalDuration ≥ e.
                    ↪ chosenTravel.totalDuration)
}

fact{
        all e: LocalEvent | (allFalse[e] = True and User.preference.cheapest = True
```

```
            ↪ ) implies
                (all t: Travel | t in e.travel implies t.totalCost ≥ e.chosenTravel
                    ↪ .totalCost)
}

//if the user checks the disability option, means without disability support are
    ↪ not considered to find the best way to travel
fact{
        User.disability = True implies (all t: Travel | all s: t.stretch | s.
            ↪ transport.disabilitySupport = True)
}

//avoid overflow
fact{
        all e1, e2: Event | add[e1.endTime, e2.travelTime] > 0
        all e1, e2: Event | e1 in FlexibleEvent implies add[e1.effectiveEndTime, e2
            ↪ .travelTime] > 0
        all e: FlexibleEvent | sub[e.effectiveStartTime, e.travelTime] > 0
}

//the events must not overlap each other and must be separated at least by the
    ↪ travel time
fact{
        all disj e1, e2: Event | (e1 not in LastingEvent and e2 not in LastingEvent
            ↪ ) implies ((e1.startTime ≤ e2.startTime and e2.startTime < add[e1.
            ↪ endTime, e2.travelTime]) implies (e1 in FlexibleEvent or e2 in
            ↪ FlexibleEvent))
        all disj e1, e2: Event | (e1 not in LastingEvent and e2 not in LastingEvent
            ↪ ) implies ((e1.startTime ≤ e2.startTime and e2.startTime < e1.
            ↪ endTime) implies (e1 in FlexibleEvent or e2 in FlexibleEvent))
        all disj e1, e2: FlexibleEvent | (e1.effectiveStartTime ≤ e2.
            ↪ effectiveStartTime implies e2.effectiveStartTime > add[e1.
            ↪ effectiveEndTime, e2.travelTime])
        all disj e1, e2: FlexibleEvent | (e1.effectiveStartTime ≤ e2.
            ↪ effectiveStartTime implies e2.effectiveStartTime > e1.
            ↪ effectiveEndTime)
        all f: FlexibleEvent | (all e: Event | (( f.startTime ≤ e.startTime and e.
            ↪ startTime < f.endTime) implies ((f.effectiveEndTime < e.startTime)
            ↪ or e in LastingEvent or e in FlexibleEvent)))
        all f: FlexibleEvent | (all e: Event | (f.startTime ≤ e.startTime and e.
            ↪ startTime < add[e.travelTime, f.endTime]) implies ((f.
            ↪ effectiveEndTime < add[e.travelTime, e.startTime]) or e in
            ↪ LastingEvent or e in FlexibleEvent))

        all f: FlexibleEvent | all e: Event | (e.startTime ≤ f.startTime and f.
            ↪ startTime < e.endTime) implies ((f.effectiveStartTime > e.endTime)
            ↪ or e in LastingEvent or e in FlexibleEvent)
        all f: FlexibleEvent | all e: Event |( e.startTime ≤ f.startTime and f.
            ↪ startTime < e.endTime) implies ((f.effectiveStartTime > add[f.
            ↪ travelTime, e.endTime]) or e in LastingEvent or e in FlexibleEvent)
}

//the travel time for each local event is equivalent to the evaluated travel time
fact{
        all e: LocalEvent | e.travelTime = e.chosenTravel.totalDuration
}

//once the Go button is tapped, the evaluated travel mustn't overlap other travels
    ↪ or events with the needed presence
fact{
        all g: LetsGo | all i: Int| (i > g.letsGoTime and i < add[g.letsGoTime, g.
            ↪ chosenTravel.totalDuration]) implies ((no s: StandardEvent | sub[s.
            ↪ startTime, s.chosenTravel.totalDuration] > i and s.endTime < g.
            ↪ chosenTravel.totalDuration) /*no overlap with standard events*/ and
            ↪ (no t: TransferEvent | t.startTime > i and t.endTime < g.
            ↪ chosenTravel.totalDuration) /*no overlap with transfer events*/ and
            ↪ (no f: FlexibleEvent | sub[f.effectiveStartTime, f.chosenTravel.
            ↪ totalDuration] > i and f.endTime < g.chosenTravel.totalDuration)) /*
            ↪ no overlap with flexible events*/
        //there is no overlap between the Go button evaluated travels
        all disj g1, g2: LetsGo | g1.letsGoTime ≤ g2.letsGoTime implies (no i: Int
```

```
            ↪ | i ≥ g2.letsGoTime and i ≤ add[g1.letsGoTime, g1.chosenTravel.
            ↪ totalDuration])
}

fact{
        all e: LocalEvent | e.startTime ≥ e.travelTime
}

//the travels cost must not exceed the maximum expence decided by the user
fact{
        all e: LocalEvent | e.chosenTravel.totalCost ≤ User.maxCostVal
}

//max foot distance
fact{
        all t: Travel | all f: Foot | f in t.foot iff f in t.transport
        all t: Travel | computeFootLength[t.foot] ≤ User.maxFootVal
}

//limitation in the alloy sums requires stretch differentiations
fact{
        all e: LocalEvent | all t: e.travel | (all disj s1, s2: t.stretch | ((s1.
            ↪ duration ≠ s2.duration) and (s1.length ≠ s2.length) and
                                (s1.cost ≠ s2.cost or s1.cost = 0) and (s1.CFP ≠ s2
                                    ↪ .CFP or s1.CFP = 0)))
}

fact{
        all disj s1, s2: Stretch | (s1.startPosition = s2.startPosition and s1.
            ↪ endPosition = s2.endPosition) implies s1.transport ≠ s2.transport
}

//if required, the user will not be advised to take public transports in the night
    ↪ time (before 6 a.m.)
fact{
        User.safePublicTimeRide = True implies (all s: Stretch | s.transport in
            ↪ Public implies (no i: Int | (i ≥ s.startTime and i ≤ s.endTime and
            ↪ rem[i, 24] < 6)))
}

//if required, the user will not be advised to ride bike when the weather is
    ↪ supposed to be rainy
fact{
        User.rainPrevention = True implies (all s: Stretch | s.transport in Bike
            ↪ implies (no r: Rain | (r.startingTime =< s.endTime and r.endingTime
            ↪ ≥ s.startTime)))
}

//the travels are designed to end at the event starting time
fact{
        all s: StandardEvent | all t: s.travel | (t.lastStretch.endTime = s.
            ↪ startTime)
        all f: FlexibleEvent | all t: f.travel | (t.lastStretch.endTime = f.
            ↪ effectiveStartTime)
}

//the rainy hours mustn't overlap
fact{
        all disj r1, r2: Rain | r1.startingTime ≤ r2.startingTime implies r1.
            ↪ endingTime < r2.startingTime
}

/*************************************FUNCTION*************************************/

//returns the total travel cost
fun computeCost(stretch: set Stretch): Int{
        sum(cost[stretch])
}

//returns the carbon footprint of a stretch, considering the vehicle pollution and
    ↪ the distance
```

```
fun computeCFP(pollution: Int , distance: Int ): Int{
        mul[pollution, distance]
}

//returns the total travel CFP
fun computeTotalCFP(stretch: set Stretch): Int{
        sum(CFP[stretch])
}

//returns the travel durations
fun computeDuration(stretch: set Stretch): Int {
        sum(duration[stretch])
}

//returns true if the user doesn't set any preference for the event
fun allFalse(event: LocalEvent): Bool{
        (no event.preference.eco and no event.preference.cheapest and no event.
            ↪ preference.fastest)  implies True else False
}

//returns the total foot distance of the travel
fun computeFootLength(foot: set Travel.transport): Int{
        sum (meanDistance[foot])
}

/*************************************ASSERT*************************************/

//travels on foot don't exist if the user has the disability option on
assert DisabilityNoWalking{
        User.disability = True implies (all t: Travel | no s: Stretch|( s in t.
            ↪ stretch and s.transport = Foot))
}

//if the user has the bus pass, the stretch cost is nil
assert FreeBusWithPass{
        User.pass.busPass = True implies (all s: Stretch | s.transport = Bus
            ↪ implies s.cost = 0)
}

//if there is a night travel with a public mean, the safe ride option isn't active
assert SafePublicRide{
        all s: Stretch | (s.transport in Public and s.startTime = 2) implies User.
            ↪ safePublicTimeRide ≠ True
}

//travels by bike with the adverse weather condition option checked imply that
    ↪ there is no adverse weather at the time
assert RainPrevention{
        (User.rainPrevention = True and one s: Stretch | s.transport in Bike)
            ↪ implies (no r: Rain | one s: Stretch | s.transport in Bike and (r.
            ↪ startingTime < s .endTime and r.endingTime > s.startTime))
}

/*************************************PRED*************************************/

pred show{
        #Public > 0
        #Booked > 0
        #Owned > 0
        #StandardEvent > 0
        #TransferEvent > 0
        #FlexibleEvent > 0
        #LastingEvent > 0
        #Travel.stretch > 1
}

check DisabilityNoWalking
check FreeBusWithPass
check SafePublicRide
check RainPrevention
run show for 4 but 6 int
```

5.2 Output

Executing "Check DisabilityNoWalking"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
37159 vars. 1355 primary vars. 91147 clauses. 581ms.
No counterexample found. Assertion may be valid. 101ms.

Executing "Check FreeBusWithPass"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
37187 vars. 1352 primary vars. 91198 clauses. 626ms.
No counterexample found. Assertion may be valid. 122ms.

Executing "Check SafePublicRide"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
37182 vars. 1352 primary vars. 91172 clauses. 705ms.
No counterexample found. Assertion may be valid. 267ms.

Executing "Check RainPrevention"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
37640 vars. 1352 primary vars. 92858 clauses. 626ms.
No counterexample found. Assertion may be valid. 294ms.

Executing "Run show for 4 but 6 int"
Solver=sat4j Bitwidth=6 MaxSeq=4 SkolemDepth=1 Symmetry=20
213589 vars. 6162 primary vars. 636231 clauses. 9962ms.
Instance found. Predicate is consistent. 59647ms.

5 commands were executed. The results are:
#1: No counterexample found. DisabilityNoWalking may be valid.
#2: No counterexample found. FreeBusWithPass may be valid.
#3: No counterexample found. SafePublicRide may be valid.
#4: No counterexample found. RainPrevention may be valid.
#5: Instance found. show is consistent.

Figure 5.1: The output generated by the Alloy program

5.3 Generated world

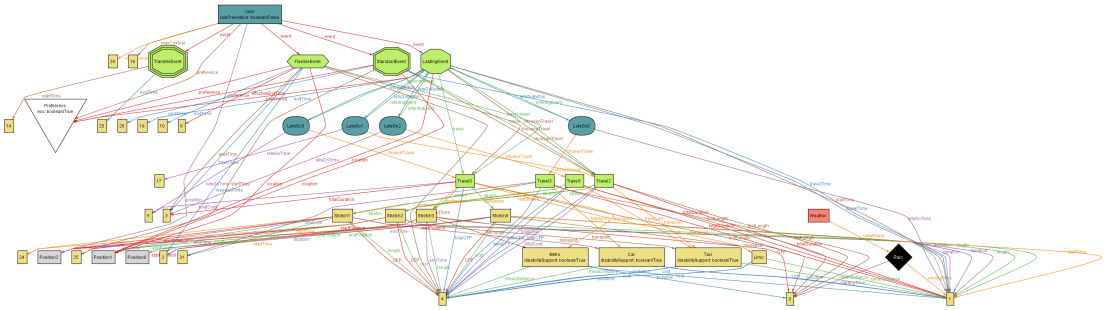


Figure 5.2: The world generated by the Alloy program

6 | Effort spent

14-oct	15:00-21:00 00:00-02:00	8 hours
15-oct	16:00-18:00 00:00-01:00	3 hours
16-oct	10:00-12:00 22:00-01:00	5 hours
17-oct	16:00-18:00 22:00-02:00	6 hours
18-oct	15:00-18:00 00:00-02:00	5 hours
19-oct	22:00-01:00	3 hours
20-oct	22:00-02:00	4 hours
21-oct	16:00-19:00 22:00-02:00	7 hours
22-oct	22:00-02:00	4 hours
23-oct	22:00-03:00	5 hours
24-oct	17:00-19:00 00:00-02:00	4 hours
25-oct	16:00-18:00 00:00-03:00	5 hours
26-oct	16:00-19:00 22:00-04:00	9 hours
27-oct	16:00-19:00 00:00-03:00	6 hours
28-oct	16:00-19:00 20:00-00:00	7 hours
29-oct	10:00-13:00 15:00-23:00	11 hours