POLITECNICO MILANO 1863

TRAVLENDAR+ | SOFTWARE ENGINEERING II PROJECT

REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENT

Authors

Antonio Frighetto

Leonardo Givoli

Hichame Yessou

Professor

Elisabetta Di Nitto

Academic year 2017/2018

Contents

Chapter 1

Introduction

1.1 Purpose

Here is a brief description of what this document, namely *Requirements Analysis and Specification Document* (RASD), aims to cover, and an initial context of the problem and its goals are illustrated thereafter.

The ultimate goal of RASD is to give an understanding of the customer requirements by describing the system itself, its functional and non-functional requirements, its components and constraints, and the relationship with the external world so that such system can be modeled with regard to the customers' needs. It is mainly addressed to project managers, systems analysts, developers, testers, and can be useful to final users as well. Being legally binding, it may be used in a contractual requirement.

Travlendar+ is an intelligent calendar-based time-aware cross-platform application, whose goal is to simplify and augment the way the final user is used to handling its own events and appointments. Users will feel free to create as many meetings as they may need, and it will be up to the application to manage them by arranging such appointments according to the travel time and his or her position so as to ensure that he or she is not going to ever be late. This implies that the application must recognize what kind of transport mean is best to reach specific locations, be it public or private, shared or not. Should it be a public mobility option and tickets purchase be available for that city, then the user may be able to buy them directly within the application. Likewise, should a bike sharing system exist nearby, then it would be surely prompted to the user - provided that he would arrive to the destination early enough. In any case, while the application will possibly suggest the best transport mean (or a combination) of them to the user, he or she can eventually choose the one that suits him or her best, according to his or her needs or wishes. Besides the aforementioned features, the application is provided with an additional module:

Lunchtime event: it allows the user to insert an event for lunch at whatever time he or she wants, as long as it fall into a range of hours, and events will be rearranged accordingly.

Let us now point out the main goals in more detail. From here on, the term *users* clearly refers to registered users.

- [G0] Allow unregistered user to sign up to access to the application.
- [G1] Allow registered users to log in and access the application and start viewing their appointments.

- [G2] Allow users to create meetings by entering a title, date and time, and the location of the event, and optionally, a description, and the type of event.
- [G3] Allow users to select a maximum time interval for a certain event.
- [G4] Allow users to modify and delete newly created events.
- [G5] Give users a friendly overview of the day's events and the scheduled time to reach a location alongside the travel mean to be used.
- [G6] Show users public and private transportations (or a combination of them) available for a specific trip.
  - [G6.1] Provided results must have been undergone to a selection which may depend on unavailability of specific transport for a particular day or, for instance, climate conditions.
  - [G6.2] Prevent users from suggesting transports which are, say, out of scope (if two locations are pretty near, then they can be reached on foot, or after a certain hour a transport may be suspended).
  - [G6.3] Allow users to deactivate specific settings, such as driving (in case of inability to drive), tolls (in case of willingness to free-only routes), or enable settings that may use only sustainable transports.
  - [G6.4] Allow users to benefit from results found, like sharing systems or own vehicles, if provided by the user.
- [G7] Give users the chance to customize the travel mean by themselves, after suggesting the best mobility options that can be taken.
- [G8] If enabled, alert users whether a specific locations is not reachable in the expected time.
- [G9] If enabled, alert users whether a specific meeting is taking longer than expected.
- [G10] Allow users to buy in-app tickets for public transportation or book a taxi.
- [G11] Allow users to insert events with flexible time occupation (lunch).

1.2 Scope

There exist phenomena that occur in reality but cannot be observable by the application, and phenomena which are controlled by the system and are unreachable from the outside. The connection between the world and the machine is made possible by the shared phenomena.

Here we include an analysis of the world and of the shared phenomena.

World phenomena

- User is physically unable to open and use the application.
- User's appointment takes longer than expected.
- User accidentally messes up or a glitch takes place.

- User's own vehicles run out of gasoline.
- A shared bicycle damages.
- Public transport goes out of work.
- Weather unexpectedly changes.
- Location becomes unreachable.
- Tickets are sold out.

Shared phenomena

- User logs into the system.
- User's appointment creation.
- Warn if appointment is getting closer.
- User buys a bus (or train) ticket or books a taxi.
- Following an accident, appointments are rearranged.
- Public transports changes are reported in real time.
- Potential weather changing is reported in time.

1.3 Definitions

RASD: Requirements Analysis and Specification Document (this document).

System: the whole software system to be developed, comprehensive of all its parts and modules. System and application are often used interchangeably.

User: any user that downloads the application and, after registering, starts using it.

Calendar-based: what it shows up is a timeline for the current day or a calendar grid.

Time-aware: it automatically recognizes when specific events occur, prevent events from overlapping among each other.

Cross-platform: multiple computing platforms are supported.

API: Application Programming Interface.

1.4 References

This document follows the IEEE 29148:2011[1] for the requirements engineering for systems and software products and IEEE 830-1998[2] for the recommended practice for software requirements specifications.
It is based on the specifications document of the RASD assignment[3].

1.5 Overview

This document is structured as follows:

Chapter 1: Introduction. It provides a general description of the system purpose and scope, along with some information about this document.

Chapter 2: Overall Description. It provides the underlying perspective over the principal system features, constraints and those factors which affects the product.

Chapter 3: Specific Requirements. It specifies thoroughly functional requirements, with the use case diagrams, descriptions, sequential diagrams, and nonfunctional ones.

Chapter 4: Alloy Analysis. It provides the Alloy[4] model for this application.

We conclude with a **Bibliography** and an **Appendix**, which consists of the software and tools used, and hours of work per each team member.

Chapter 2

Overall Description

2.1 Product perspective

The application will be available through web and mobile devices, keeping the same look and elements along the various implementations. The user experience through the application should be easy and natural.
The application developed allow the time slotting ahead the upcoming events so that the user can have a full view of the day with a prediction of the travel time between the appointments if they are in different locations. The event can regard working or personal aspects, and in any case, the application will notify or suggest a different mean of transport based on the weather conditions or strikes of the public transport.

2.2 Product functions

The application allows you to schedule events, choose which mean of transport and get warned in case of overlapping events On top of the time allocation, there will be a cost estimation provided when possible. This will allow the user to filter the possible mean of transport based on carbon footprint emission, price impact and time.
A User can:

- Create an event:
  - Modify an event
  - Invite other users to their events
- Receive help to reach the location of the next event
  - Buying tickets for a mean of transport that require tickets
  - Providing navigation for reaching the spot
- Get notified
  - Upcoming events
  - Free space

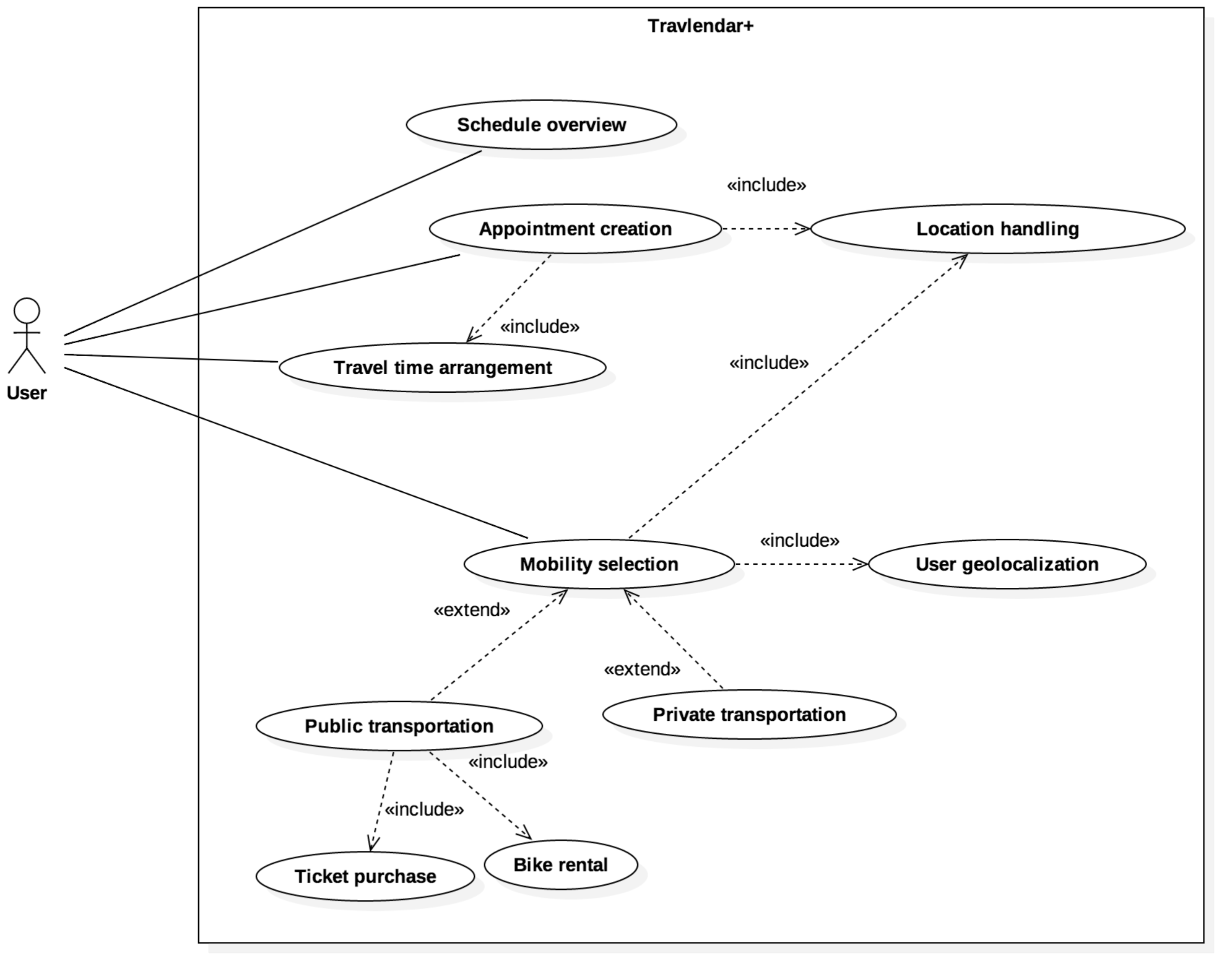


Figure 2.1: Comprehensive use case diagram.

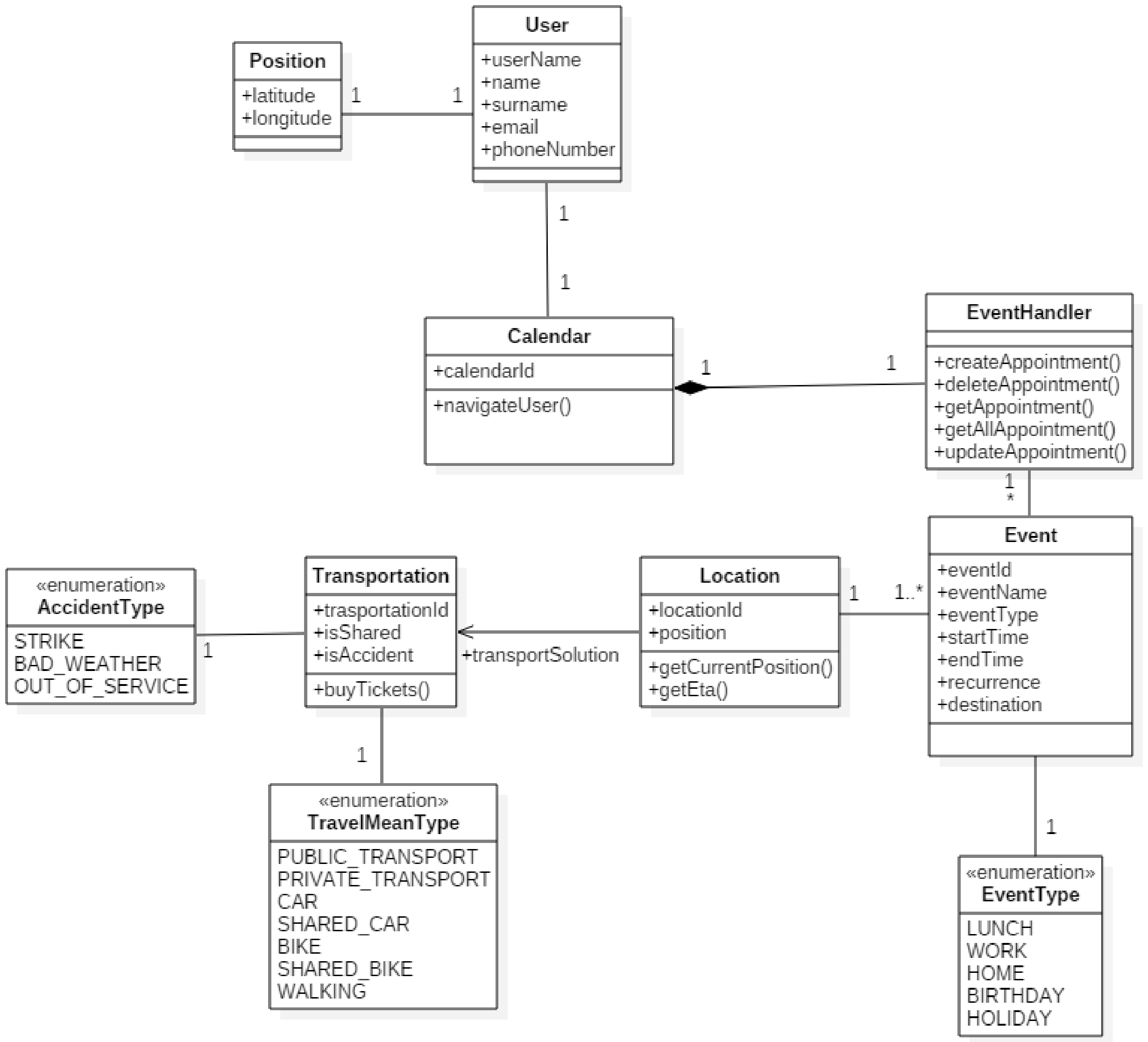


Figure 2.2: Comprehensive class diagram.

2.3 User characteristics

There is one type of user, which can be interfaced with other users in case of shared meetings.

Instead, the users that receive the invite will have read-only permissions.

2.4 Assumptions, dependencies and constraints

Text assumptions

- The user will be able to schedule only future events, with the option to recur on a daily, weekly or monthly basis.
- The working events can only be scheduled within the working hours, reserving at least 30 minutes for the lunch break.

Domain assumptions

- [D1] Two events can be overlapping but cannot start and end at the same time (for instance, one event occurs throughout the day).
- [D2] The event that overlap the other cannot have an end time previous the event started before.
- [D3] The event located within the same location have no mobility option (navigation/buying tickets).
- [D4] Working events can be scheduled within the working hours.
- [D5] The user have to configure his habits (working hours, owning a car/bike) at the first use of the application.
- [D6] The usage of public transportation is recommended within its working hours, in our case 6-24.
- [D7] When the location of 2 events is further than 1.5km, the system won't suggest walking.

Dependencies

We will depend on Google Maps API's to get the estimation of the time to get the distance between 2 locations. We will access the Google Maps Distance Matrix API through an HTTP interface, with requests constructed as a URL string, using origins and destinations.

During the process to buy the tickets of the public transport system we will redirect the user to the provider of the service, or when allowed the user will be deep-linked to the desired application.

Constraints

To ensure enough level of privacy and avoid security breaches, every request containing the user's position will be handled within an HTTPS connection.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

3.1.1 User interfaces

The User Interface in every implementation should be intuitive and give the user the perception of control across the functionalities given from the application. The application has to support multiple languages, with a centralized label manager. This should be enforced by the coherence of all the functionalities across every the platform. Besides that, every platform must have a User Interface compliant with the latest major commercial standards available:

- Mobile
  - The application will be developed with Xamarin.Forms and so the UI will be shared across the iOS and Android implementations. On top of that, the 2 versions must follow the iOS Human Interface Guidelines and the Android Material Design Guidelines.
  - Due to the huge variety of the Android devices, the application will provide the pixel perfect compliance relying on the 2 major screen sizes (normal, large) and 4 different densities (ldpi, mdpi, hdpi, xhdpi) depending on Android OS to auto-scale the missing ones.
- Web
  - The web application must be compliant with the W3C standards, concerning HTML and CSS. The UI should be modular and independent from business logic, supporting the major browsers (IE, Edge, Chrome, Firefox, and Safari).

3.1.2 Hardware interfaces

The application will require the authorization to access the user's location, available through the GPS and/or the WiFi/Data connection.

3.1.3 Software interfaces

The clients will implement the core functionalities interfacing with the Google Maps Distance Matrix API, Yahoo Weather and whenever possible the API exposed by the shared transport systems chosen by the user. Saving user's data will be managed through SQLite, with built-in hooks to encrypt them.

3.1.4 Communication interfaces

The application will be based on a service integration layer built upon REST API allowing all the different implementation to retrieve consistent information. This will allow the platform to ensure the scalability of the number of user and new features.

3.2 Functional Requirements

The following are the functional requirements of the software, extracted from our analysis, concerning each actors of the system. Each of them consists of some scenarios and related use case, sequential and statechart diagrams.

User Login and Registration

Purpose

Any user is encouraged to subscribe through the web application or the mobile one. The system provides the user the possibility to become a registered user by filling a registration form or directly accessing through third-party accounts like Google or Facebook. After authenticating, functionalities concerning the account management are also provided, so the user can easily:

1. Login into his/her account.
2. Recover forgotten password by resetting it
3. Update account data.

Users are simply asked to insert this information:

- E-mail address
- Username
- Password
- Name
- Surname

Some basic checks are performed when the user is asked to fill the form in order to register. For instance, the system requires the user to insert a password that contains at least one number, one capital letter and a symbol, whose length is not less than 8 characters. The user will be asked to insert the password twice. Clearly, if the privacy policy and terms of condition are not authorized, the subscription is nullified. The user is free to delete his account at any time. Credentials are encrypted stored in the device so that the user must never insert them but the first time. Whilst asking for registration and logging in to access the application can be seen as a waste of time, it is designed for allowing the user to manage his/her own reminders and events both from the mobile and the web application so that they are always synchronized.

Scenario 1

Alice decides to give the application a try, so, after downloading and launching it, she is displayed a login page where she is asked to sign up in order to access to all its services. She immediately notices the Google logo and, to avoid wasting time by creating another account, she decides to sign up with Google. Everything goes well, and she is welcomed by the user-friendly interface and the application is now fully working.

Name	Login
Goal	G1
Actors	Registered User
Assumptions	• The user has already signed up into the system. • The user is not logged into the system yet.
Events flow	1. The user opens the *login page* of the system; 2. The user types in username and password. 3. The system recognizes the identity and ensures that the user who is logging it is who he claims to be. 4. The user can visualize his personal calendar and access to the system's functionalities provided to him.
Exit conditions	The user is now logged into the system.
Exceptions	The *username and password* inserted are wrong, an error message is shown. The user is not logged.

Table 3.1: Use case for user login.

Scenario 2

Bob accesses Travlendar+ application through the web page and clicks on the *Login* button. He is asked to enter his username and password, but figures out that he had never joined to the service before, so he goes back and decides to sign up. Being completely new, he creates a totally new account. When filling the form, he inputs *Bob* as username. Turns out that this username has already been used so Bob is warned that that username is unavailable. He changes username, continues fulfilling the form and eventually, a window welcomes him by starting a little demo which guides him around the website. He is also informed that to fully access the application he must confirm the evidence of the registration by clicking the link sent via email.

Use case

The use case for user registration is shown in Table 3.1, whereas the use case for user login is shown in Table 3.2.

Sequence diagram

The sequence diagram of the login process is illustrated in Figure 3.1.

Statechart

The statechart of the registration process is illustrated in Figure 3.2.

Name	Registration
Actor	Unregistered user
Goal	G0
Input condition	The user creates a new user account or signs up with third-party accounts.
Events flow	1. The user decides whether he/she wants to create a completely new account or to undergo an application-based enrollment. 2. Either the user is redirected to the third-party sign-up page and asked to insert his/her credentials or a registration form is loaded and the user is asked to compile it. 3. In both cases, the user is asked to authorize the privacy policy and terms of conditions. 4. If the user decided to create a new account, then he/she can confirm the registration by accepting the linked sent via email. No two-factor authentication is called for.
Output condition	The system welcomes the user by informing him/her that the registration was done successfully.
Exception	• If username or similar data are already been taken or invalid username is provided, the user is warned to choose for a different username. • If no account exists when signing up through third-party services, they will also handle resulting possible errors.

Table 3.2: Use case for user registration.

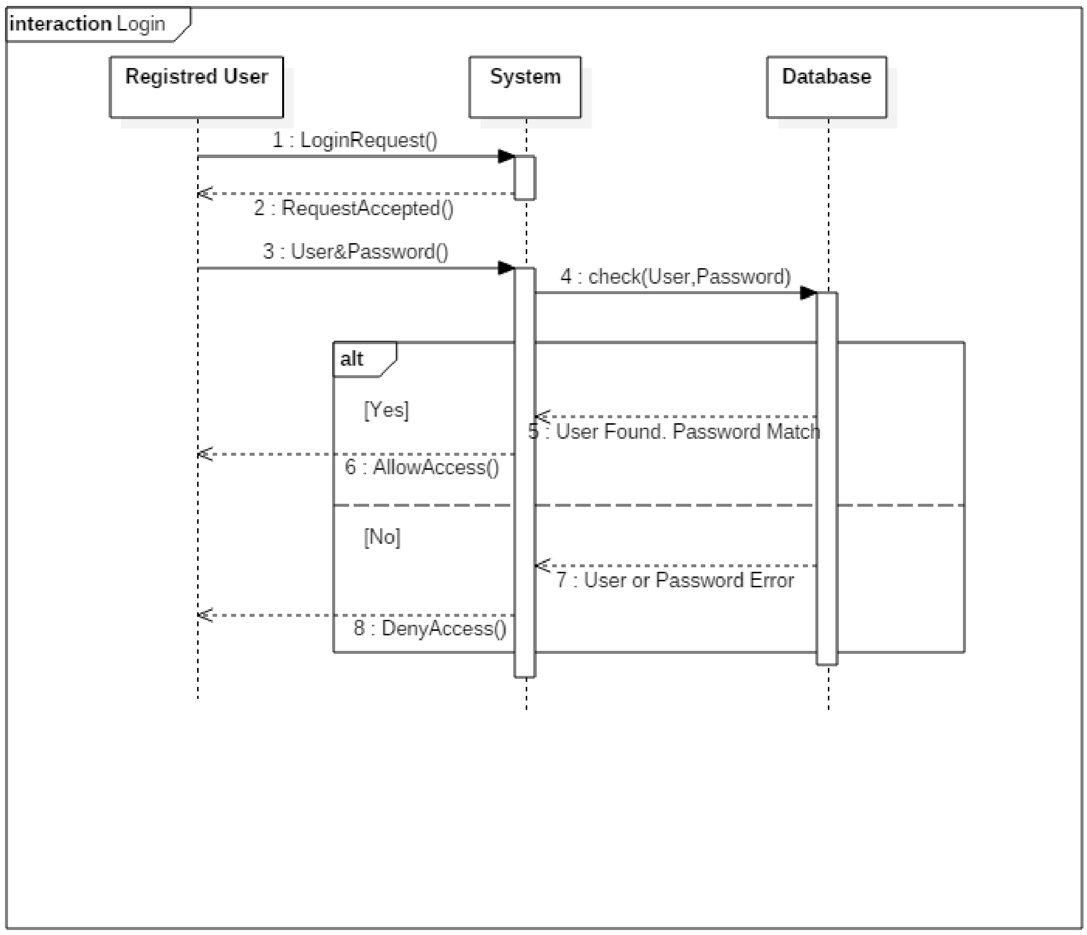


Figure 3.1: Sequence Diagram: Login.

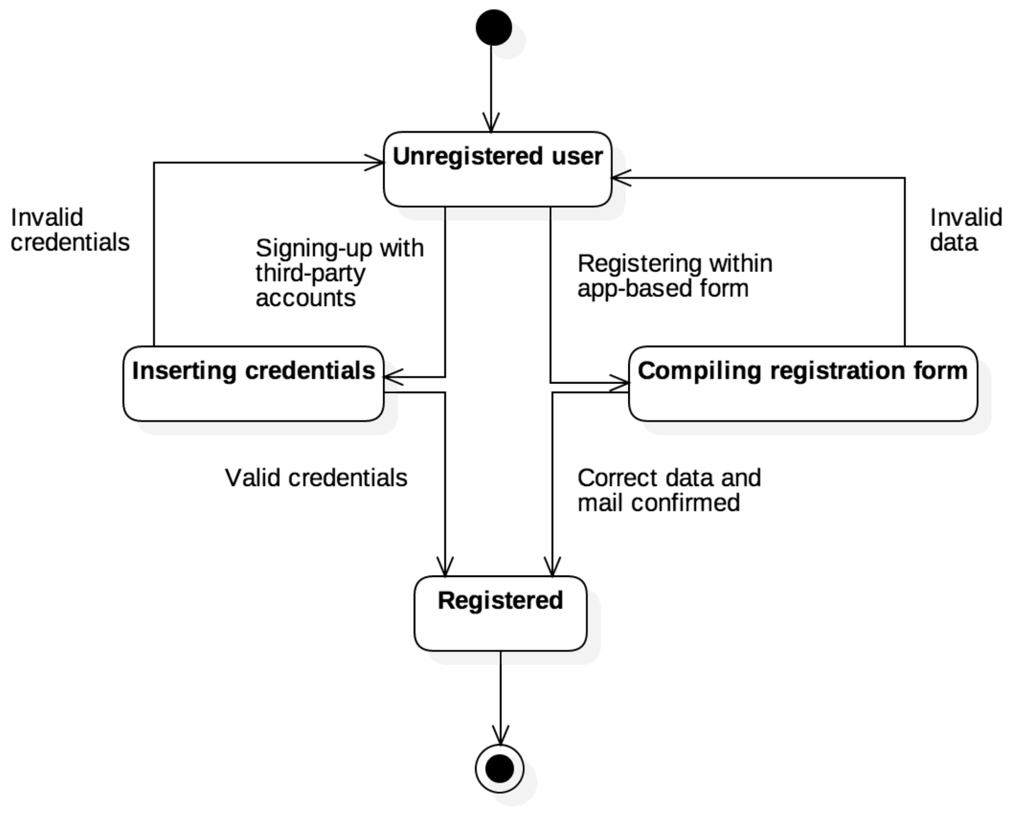

Unregistered user
Invalid credentials
Signing-up with third-party accounts
Registering within app-based form
Invalid data
Inserting credentials
Compiling registration form
Valid credentials
Correct data and mail confirmed
Registered

Figure 3.2: Statechart Diagram: Login.

Mockup

The mockup of the login interface is shown in Figure 3.3.

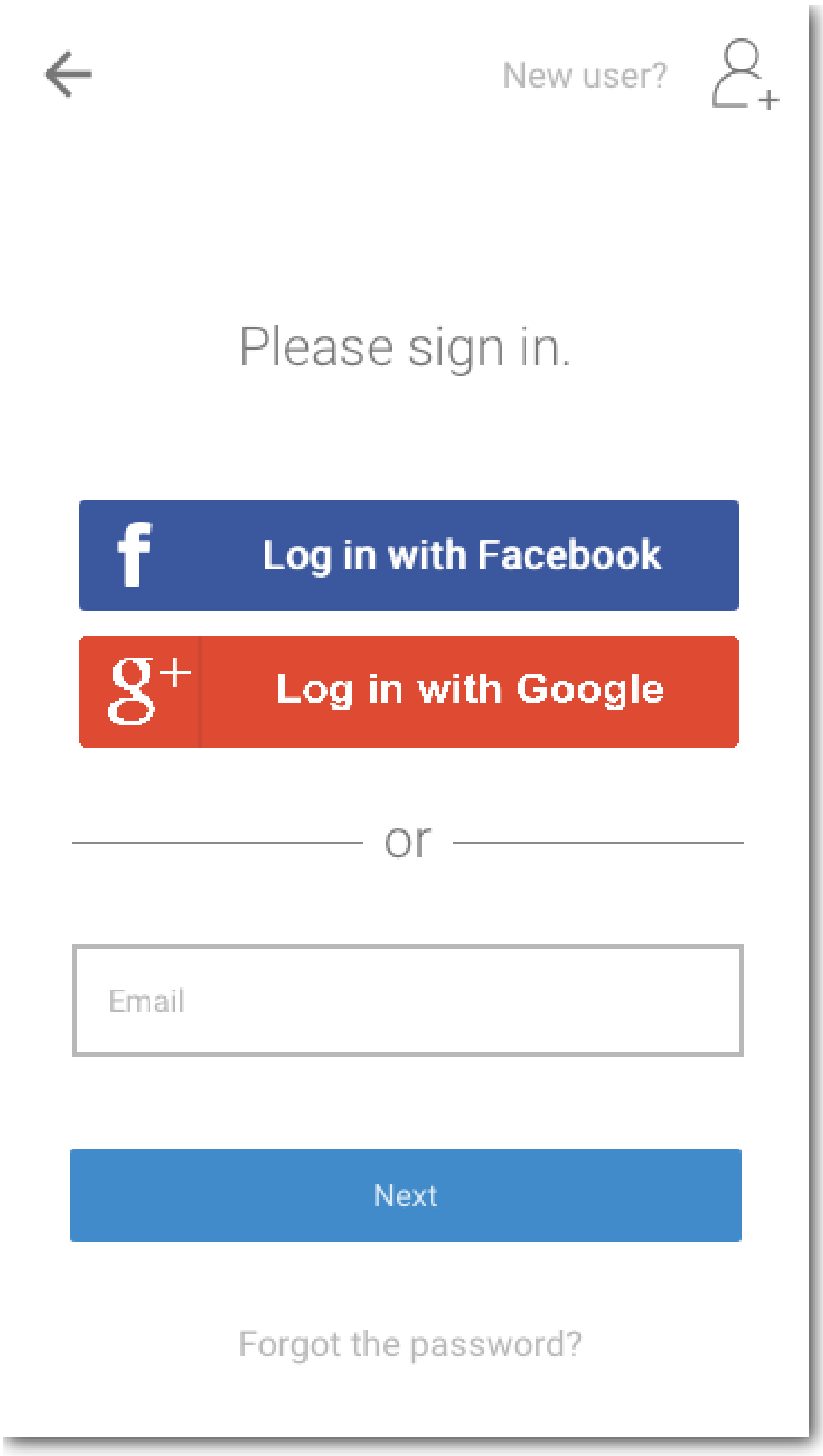


Figure 3.3: Login mockup.

Appointments overview and creation

Purpose

The user of the application who has correctly logged in the web application or the mobile one, can view and organize the appointments into his own calendar. The system provides the user the possibility to see, create, modify or delete any appointments that he/she has inserted. In particular, the main features are as follows:

- View appointments: the user is given the possibility to view all the appointments, both with the general overview in the form of a grid, and the list view with appointments and the public means interleaved.
- Create appointment: the user has to click on the specific date where he wants to add the new appointments and inserts the information below:
    1. Name of appointment;
    2. Start time;
    3. End time;
    4. Recurrence (optional);
    5. Means of transport.
- Modify appointment: the user simply clicks on the appointments that he has created so that he is able to update the information.
- Delete appointment: the user through the same gesture of *Modify appointment* can easily delete the appointment by clicking on the specific button.

Scenario 1

Alice, a young business consultant, plans all her appointments through the Travlendar+ application. Every Wednesday, the company decides to organize a refresher course. For this reason, she needs to add this appointment to her calendar. Alice through the calendar interface, clicks on the first day course and adds the necessary information to create the appointment by using the form: event name, start time, end time, recurrence, means of transport. The system checks that the new appointment meets each requirement and asks Alice to confirm. Once confirmed, the event is displayed in her list of appointments. Unfortunately, the company after a few days, deletes the refresher course, so Alice wants to delete the appointment from her calendar. Alice easily accesses the event and modifies it. By clicking the corresponding button, she deletes the event and all the saved recurrences. The system asks Alice if she is really sure to perform the operation, she confirms and the event is deleted from the calendar.

Scenario 2

Bob, after correctly signing up to Travlendar+, decides to check the appointments of the day through the calendar. While he's scrolling the appointment list, he notices that one of the places of meeting was entered incorrectly. By simply clicking on the event, Bob is able to update the information. The system checks that the updated appointment meets each requirement. After calculating the time needed to reach the place of the appointment, the system warns Bob that the travel time between the previous and the updated event is not enough to reach the place of the appointment. However, the system asks Bob if he wants to confirm or update the event. He doesn't choose to update the event, so changes are not made. Bob accesses the event and

Name	Create appointment
Actors	User
Assumption	The user needs to insert a new appointment in his/her personal calendar
Pre-Conditions	• The user has successfully signed to the system. • The user has already opened the window with the insertion form.
Flow of events	1. The user creates a new appointment by inserting all the information needed in order to adding a new event correctly in his own calendar. 2. The system checks if the appointment inserted can be successfully validated. This means that simple checks against the time (the start one should be less than the end one) and the date (should not be an expired date) are done. *Note:* overlapping events check is done at a second stage. 3. The user is informed by a warning message about the actual validation of the appointment. 4. The user have to confirm o reject the insertion of the appointment.
Post-Conditions	The appointment of the user has been stored into the system in the event that he has confirmed the inclusion.
Exception	An internal system error makes impossible to store the reservation data. The user is notified of the error

Table 3.3: Use case for create appointment.

modifies it again, but in this case, he deletes the event by clicking on the specific button. The system requires confirmation and the event is deleted form the calendar.

Use case

The use case for creating an appointment is shown in Table 3.3.

Sequence diagram

The sequence diagram for creating an appointment is illustrated in Figure 3.1.

Mockup

The mockup of the appointments overview and the list view are shown in Figure 3.5 and in Figure 3.6, the menu of the application is shown in Figure 3.7 and the creation of appointments is shown in Figure 3.8.

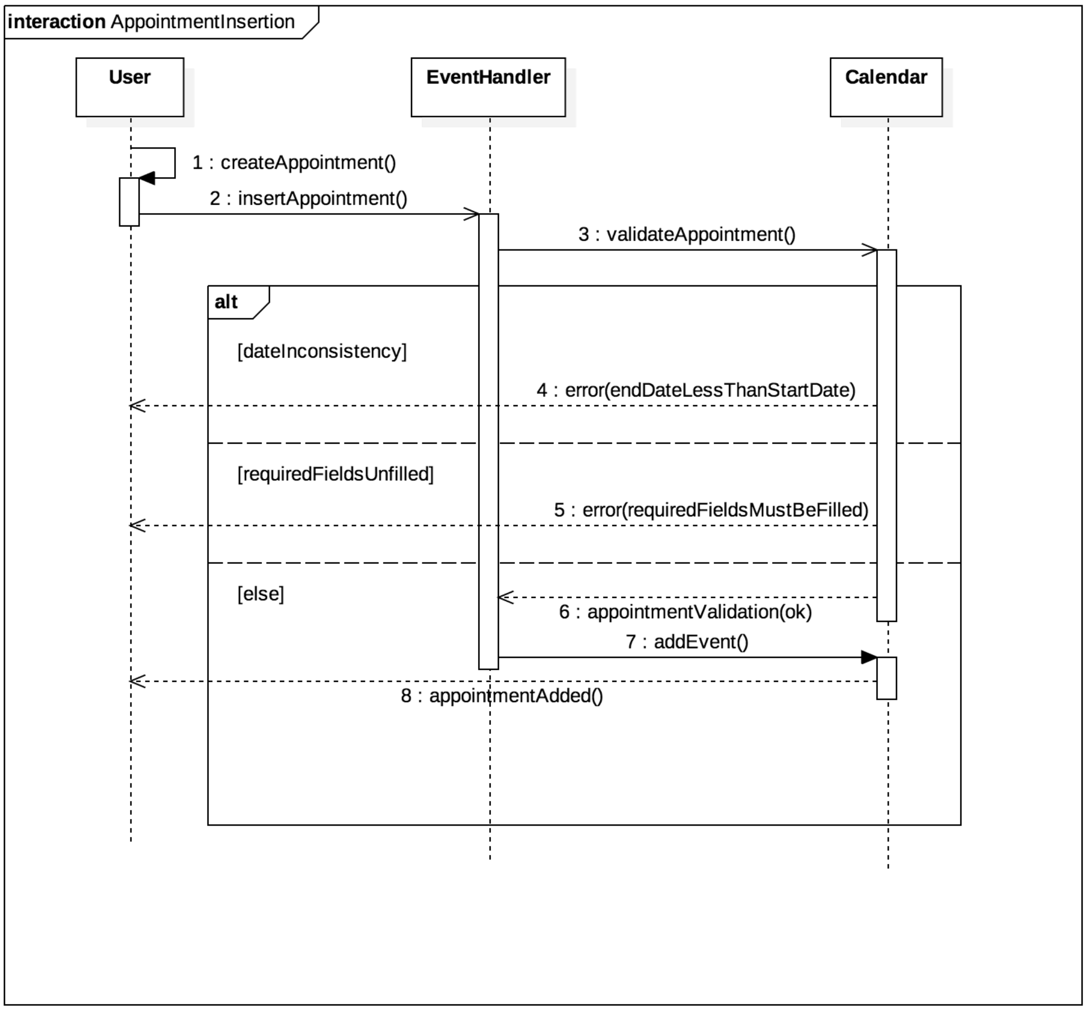

interaction AppointmentInsertion
User
EventHandler
Calendar
1 : createAppointment()
2 : insertAppointment()
3 : validateAppointment()
alt
[dateInconsistency]
4 : error(endDateLessThanStartDate)
[requiredFieldsUnfilled]
5 : error(requiredFieldsMustBeFilled)
[else]
6 : appointmentValidation(ok)
7 : addEvent()
8 : appointmentAdded()

Figure 3.4: Sequence Diagram: Create Appointment

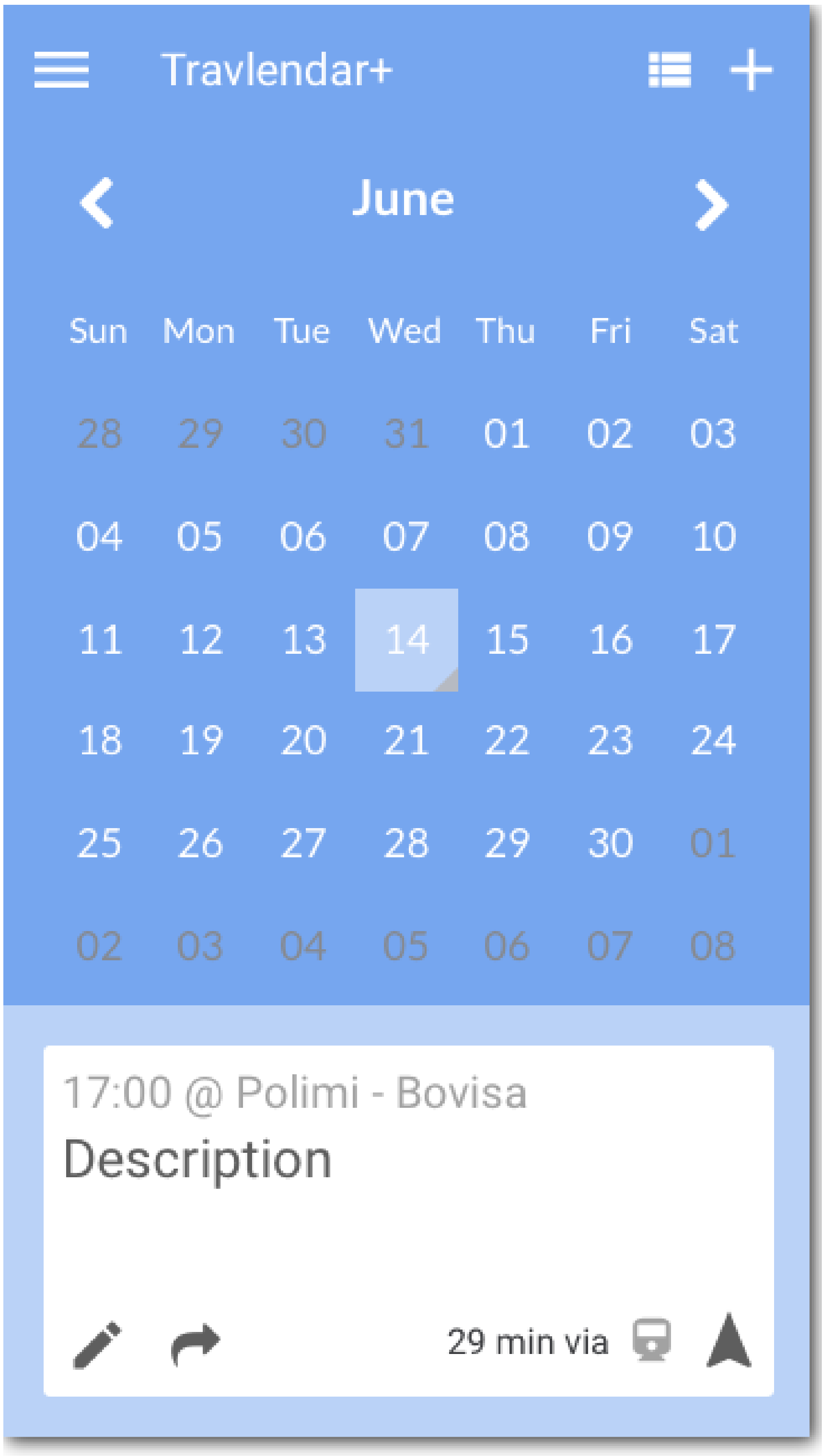


Figure 3.5: Overview appointments mockup.

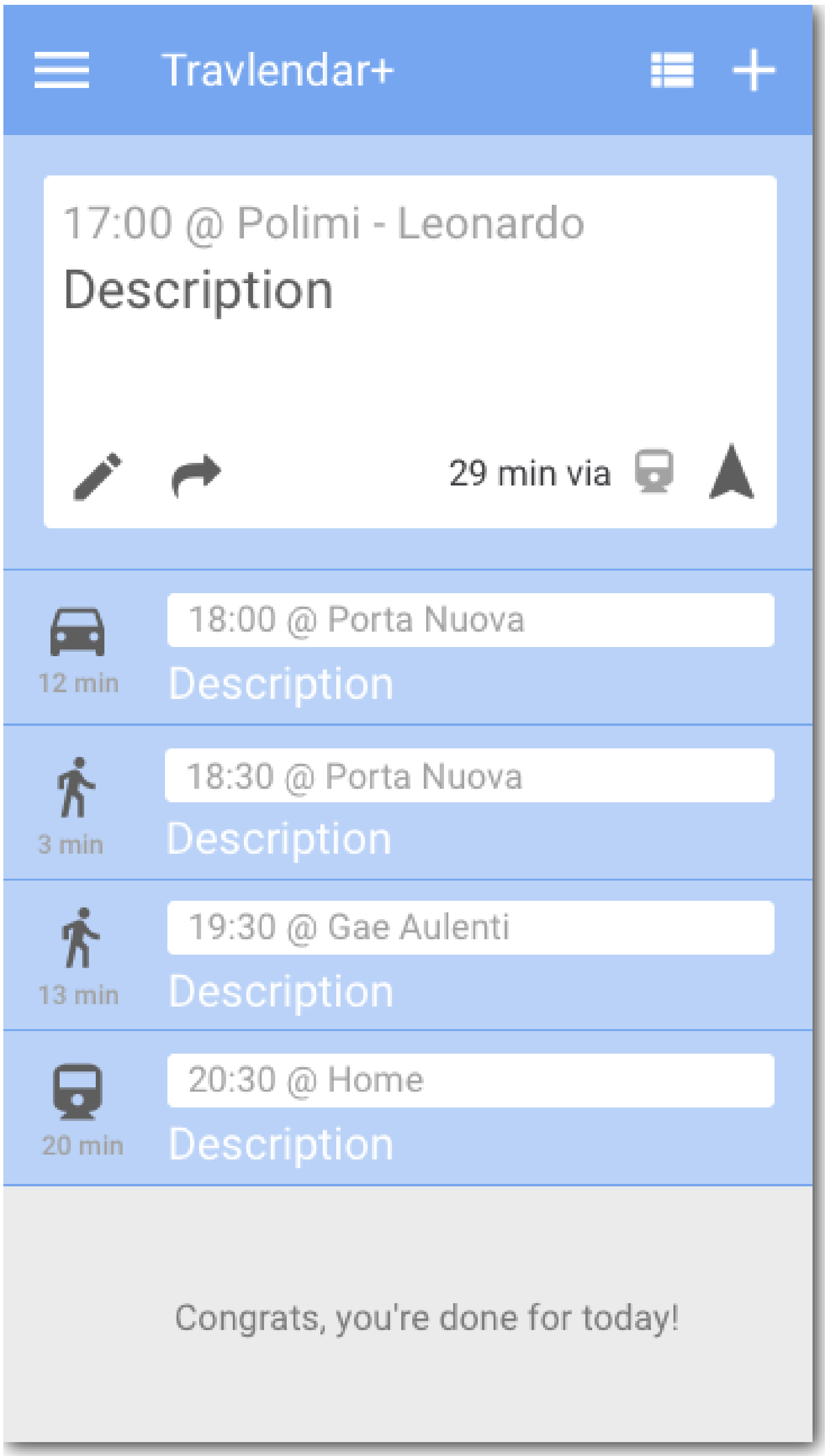


Figure 3.6: Listing view appointments mockup.

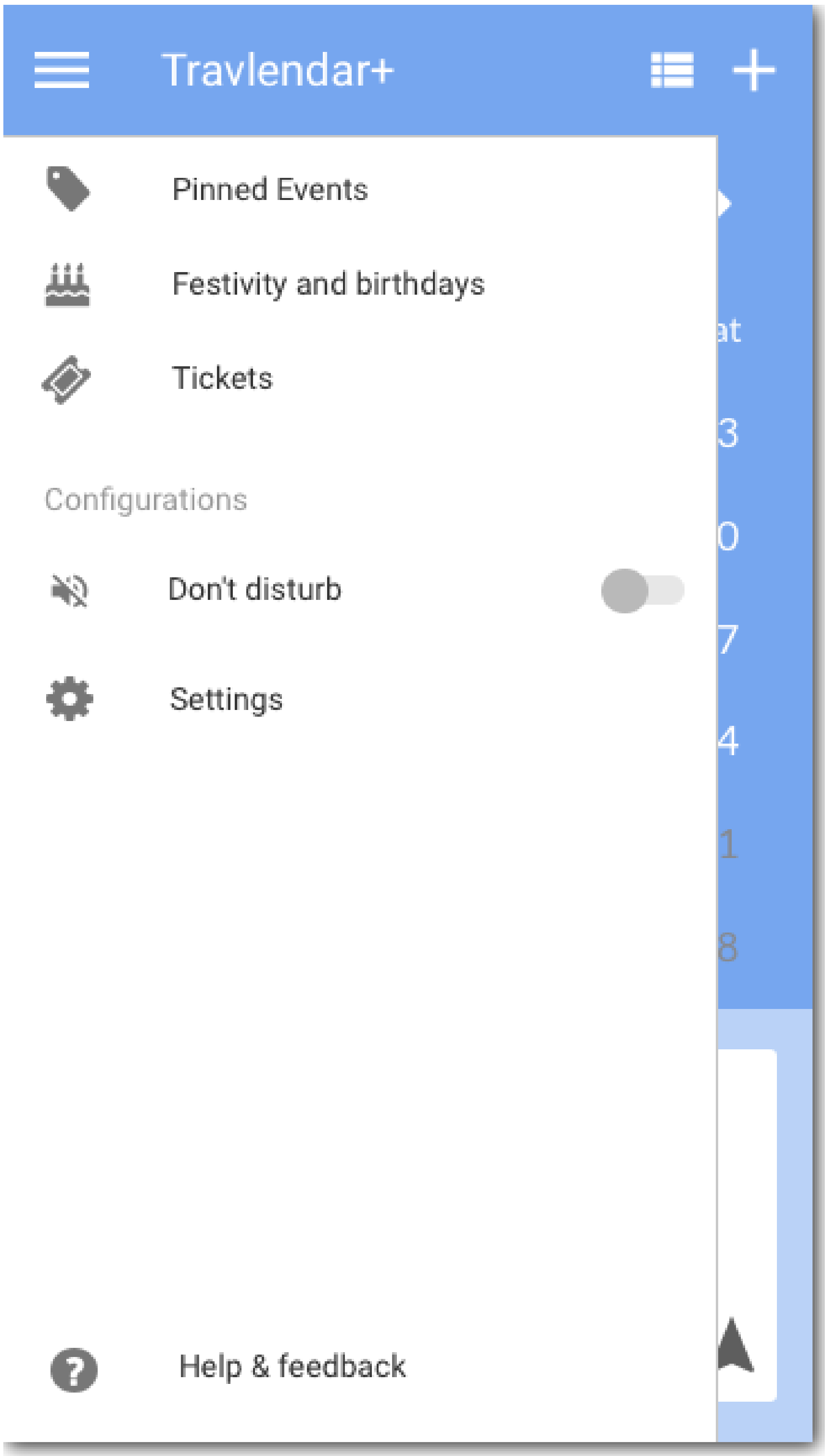


Figure 3.7: Menu view mockup.

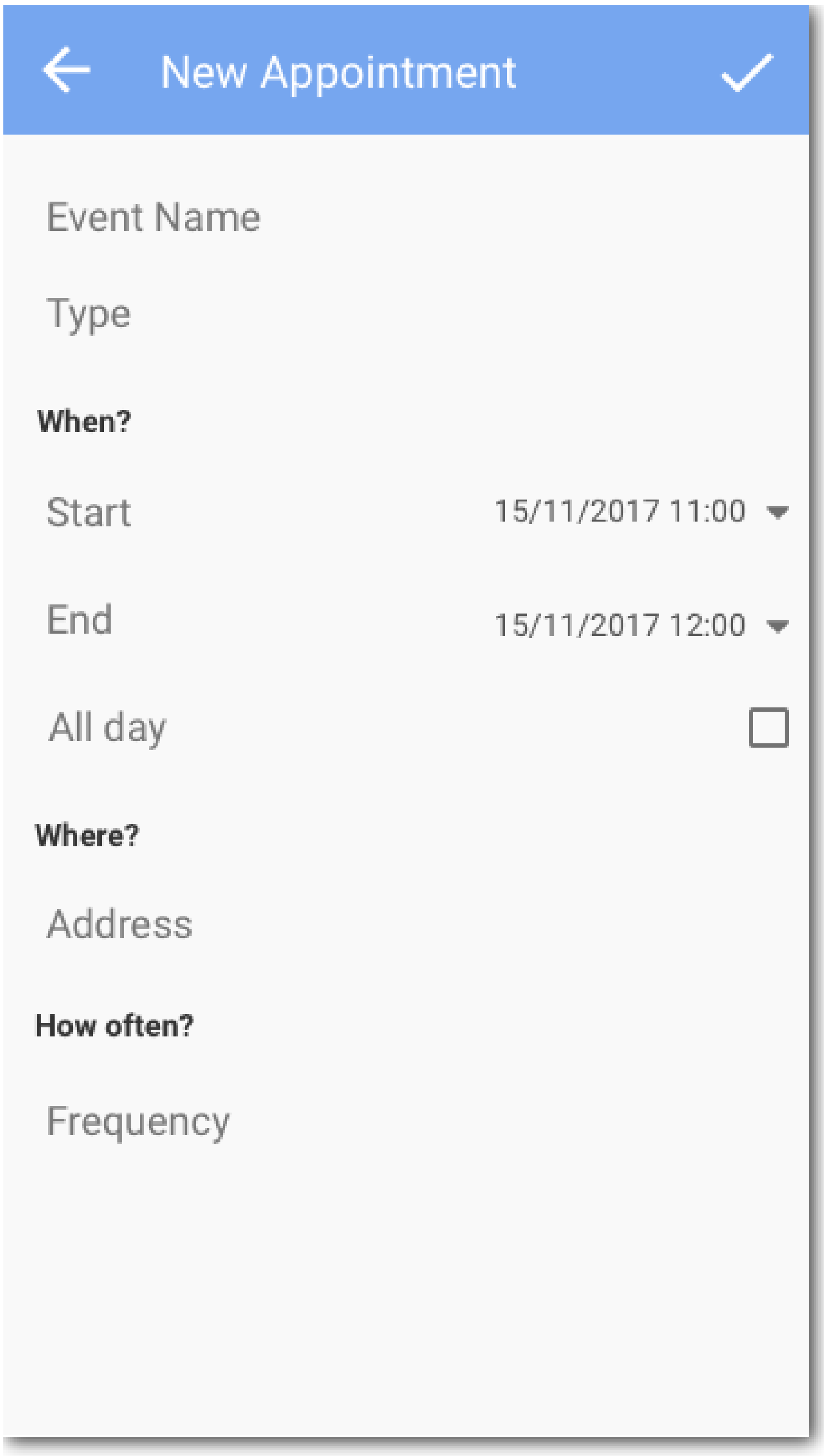


Figure 3.8: Creation appointment mockup.

Time and route travel

Purpose

A user who successfully creates events may wish to take advantage of one of the most interesting feature of Travlendar+, which concerns both time and itinerary. This makes Travlendar+ an enhanced calendar-based application. Particularly, a user can add as many events as he/she may want, and the system will be responsible to arrange time slots so as to ensure that he/she is never going to be late. The system allows the user to insert the location of the event as well so as to instruct him to take the best itinerary to reach such location. To compute the best route in terms of both space and time a sequence of calculations will be executed. Various factors - both that can be handled by the user and that which cannot - are considered.

Scenario 1

Alice is a manager who needs to go often around the city for work to meet customers. Since living in a big city and having to wake up soon, she knows that in the peak hours it's difficult to move around the city, and the system is aware of this too. She often needs to move around the entire city pretty fast. So, whenever she wants to see the task ahead, she knows she is in good hands, in fact the system typically proposes her the best route which consists of taking the bus or the underground train, or - often - a combination of them so that she can avoid the traffic of the city.

Scenario 2

Charlie, who is a little bit messy and unmindful, inserts multiple appointments for work for the current day and fills up the entire time window for which lunch is supposed to be done. Fortunately, the application reminds here not to forget to have lunch so he is prompted with a dialog to choose when he would like to insert lunchtime. Then he adds two appointments which are too close together. Itinerary is calculated anyway, but no feasible route is possible to reach the destination. The system however recognizes that 5 minutes before the end of the first appointment a bus will pass by there, so he is informed that if he desires to be in time to the upcoming event he will just have to confirm system's proposal. He accepts and a remainder for leaving earlier the first appointment is automatically set.

Scenario 3

Bob promotes conferences to sensitize environmental sustainability issues. Whenever he is invited to give lectures, he always tries to opt for travel means whose environmental impact is as low as possible in order to minimize, among other things, the carbon footprint. So he started using Travlendar+ because he is aware of this unique functionality which suggests convenient travel means for reaching various locations. Being the ecological option enabled, the system advises Bob to avoid means like cars and taxis, and, depending on how far away the destination is, the system will tell him to go on foot, or will locate the nearest bike sharing system or, at most, to use urban transports.

Scenario 4

Eve, a young girl of regular habits, who has seen an advertising of the application online, has started using it recently. At the first use, a welcome message and a little demo are shown and she is moved into another window where she is asked to set some preferences in order to fully customize the application. When checking the age, the system recognizes that she is a

Name	Travel time arrangement
Actors	System manager
Input condition	The user has already added at least one appointment and has been validated.
Flow of events	1. The user may want to create further appointments. 2. The system figures out the time window between one event and another, which means that attempts to compute whether there's enough time to get to the next appointment successfully, which is supposedly - but not necessarily - to be in a different location. 3. The system warns the user if a scheduled appointment is too adjacent to another or cannot be easily reached. 4. The system checks and rejects two appointments overlapping with each other (whose start date and time, and end date and time are identical). 5. The system ensures that at least an half an hour - at the discretion of the user to change slotted time - is reserved for lunchtime.
Output condition	The user can trust the system that all his/her appointments are properly organized.
Exception	There's no particular exception which may occur. A user, due to unforeseen circumstances, may not respect some appointments. This cannot be obviously handled by the system and no warnings are displayed.

Table 3.4: Use case for travel time arrangement

minor, so car driving and car sharing are disabled by default. The system understands that she's accustomed to inserting recurring events like going to school, so she is notified in advance if there's some traffic which may require here to leave home earlier. What's more, the system realizes that most places are reached walking - regardless of the fact that it is not that near or not - and, after a while, the system displays possible paths that can be taken on foot by default.

Use case

The use case for the travel time arrangement is shown in Table 3.4, whereas the use case for the travel route handling is shown in Table 3.5.

Sequence diagram

The sequence diagram of the handling of checks process carried out by the system is illustrated in Figure 3.9.

Name	Travel route handling
Actors	System manager
Input condition	The user has already added at least one appointment and has been validated.
Flow of events	1. A check of the user preferences is performed the very first time or whenever local settings have been changed. 2. The system controls date of event. If the event is scheduled for the current day it continues with calculating the path otherwise, otherwise it is not considered for now. 3. The system asserts that the destination location of the newly added event exists and is reachable and attempts computing the best travel path to reach the location in time. 4. The system performs specific checks against the weather condition, the fact that a bike sharing system is in the close proximity and whatnot. 5. Some confirmation dialog asking if there's a preference for a particular itinerary or public mean may show up in case multiple reasonable routes are found. 6. The system provides the user the best travel path with selected means of transport (or a combination of them) in accordance with user's preferences. 7. The user is asked to confirm, choose for a different path and following recalculation or dismiss.
Output condition	A suitable and reasonable travel path is guaranteed to be found.
Exception	There's no particular exception.

Table 3.5: Use case for travel location arrangement

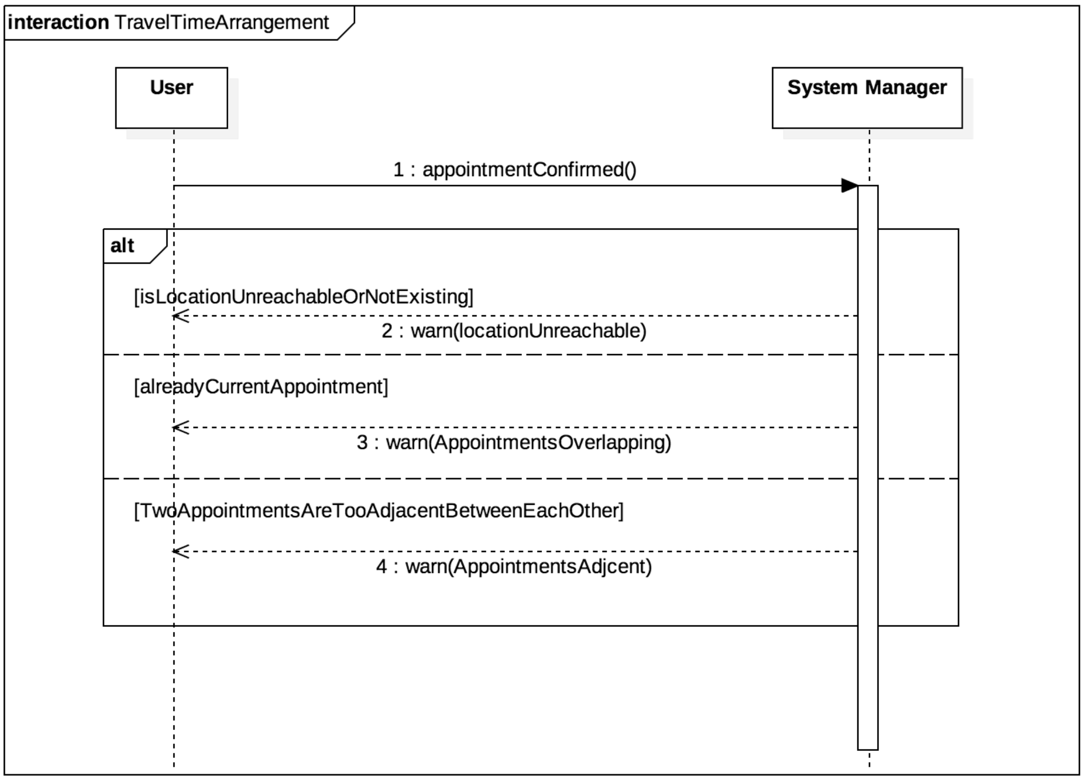

interaction TravelTimeArrangement
User
System Manager
1 : appointmentConfirmed()
alt
[isLocationUnreachableOrNotExisting]
2 : warn(locationUnreachable)
[alreadyCurrentAppointment]
3 : warn(AppointmentsOverlapping)
[TwoAppointmentsAreTooAdjacentBetweenEachOther]
4 : warn(AppointmentsAdjcent)

Figure 3.9: Sequence Diagram: Travel Time Arrangement.

Statechart

The statechart of the action undertaken by the system is illustrated in Figure 3.10.

Associated functional requirements

1. The system will ask the user the first time to authorize the use geolocalization (estimation of the current position).
2. The system calculates the best itinerary only for events for the current day. Events for the following days, after being added to the calendar, are actually evaluated only when that specific day comes.
3. The system takes into account the current position as starting point by default.
4. The system allows the user to customize travel means to be taken according to his/her preferences.
5. The system computes the best route based on both user preferences and on factors which cannot be controlled by the user (belong to the external world).
   - The system checks user preferences every time they have been changed.
   - The system disallows using car if the user is not 18-year-old yet.
   - The system will not provide a route with a specific transport, if the user has chosen not to use it.
   - The system will locate the nearest bike sharing system if the user is accustomed to riding a bike and favourable conditions occur.

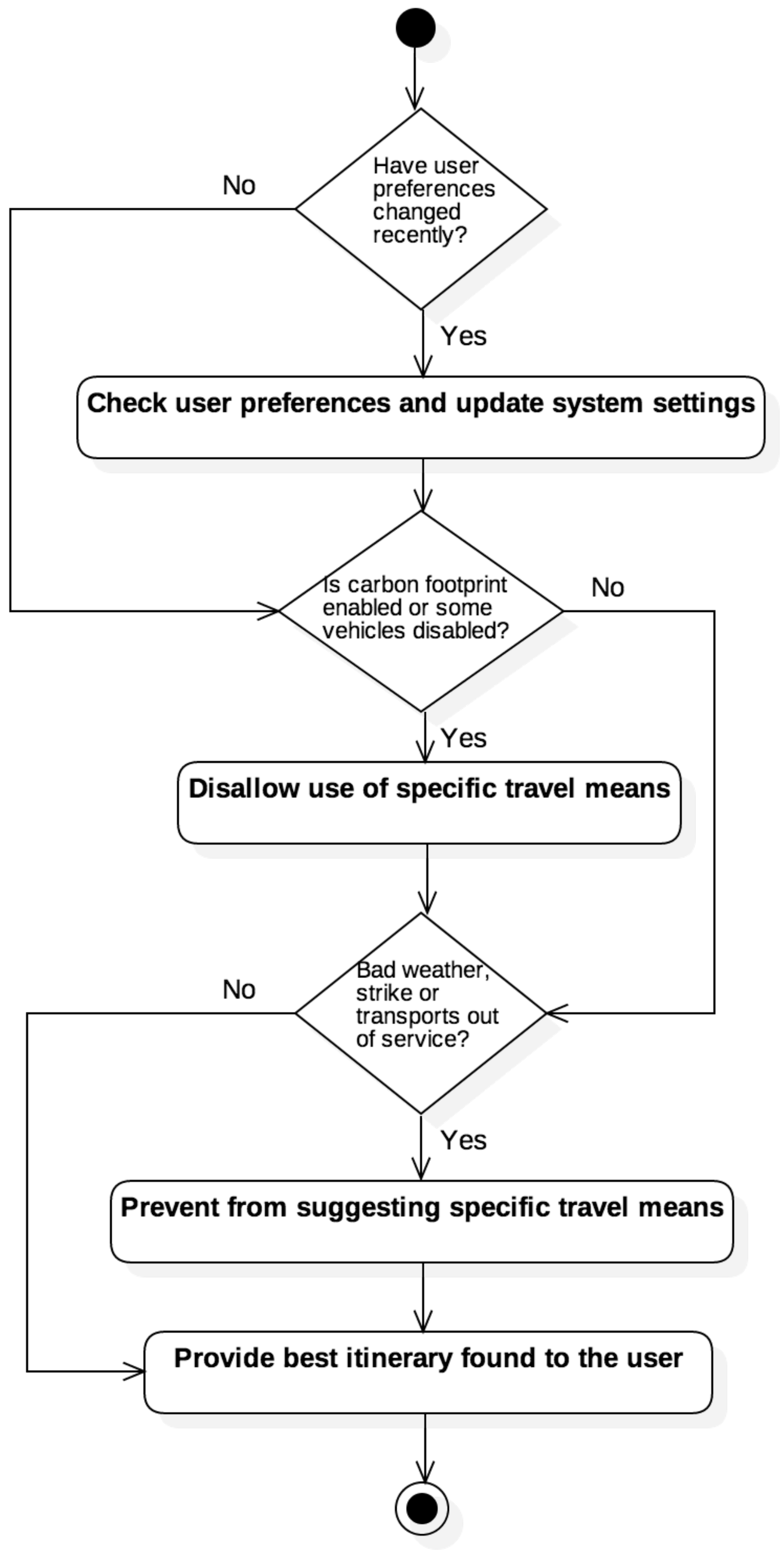

Have user preferences changed recently?
No
Yes
Check user preferences and update system settings
Is carbon footprint enabled or some vehicles disabled?
No
Yes
Disallow use of specific travel means
Bad weather, strike or transports out of service?
No
Yes
Prevent from suggesting specific travel means
Provide best itinerary found to the user

Figure 3.10: Statechart: Travel Time Arrangement.

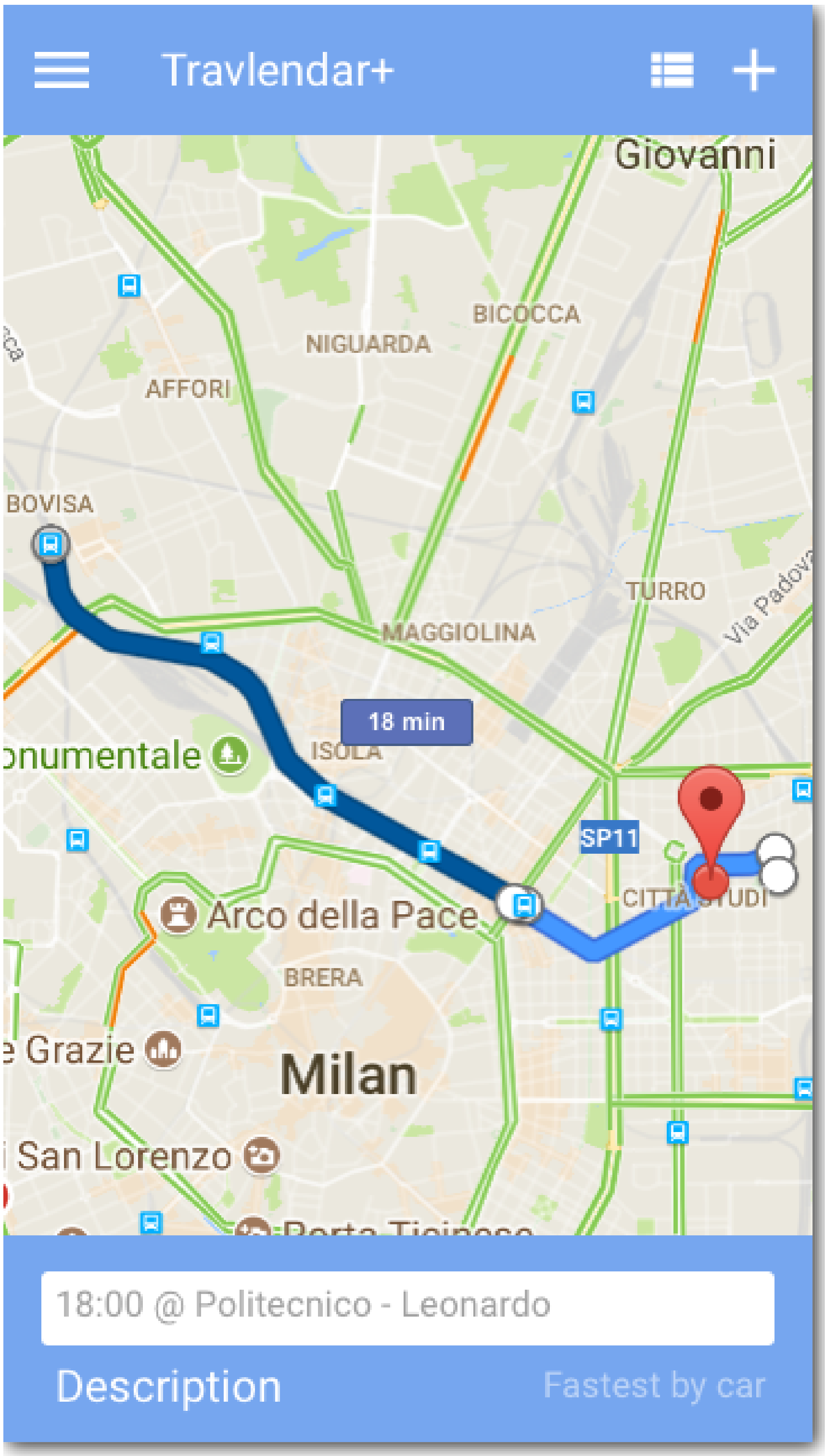
- The system will be likely to suggest walking instead of using motor vehicles if distance may take a little bit longer on foot but there's traffic jam.
- The system will unlikely suggest walking if it is going to start raining.
- The system will unlikely suggest car or taxis if the user has enabled carbon footprint optimization option.

6. The system will notify the user - when an event is approaching - that car can be taken if traffic is pretty much smooth, or underground train will be near there within few minutes, if the user is close to a station.

7. The system will reserve at least half an hour for lunchtime, or any other kind of breaks - if requested by the user.

Mockup

The mockup of the itinerary resolved with navigation with car is shown in Figure 3.11, whereas the itinerary with navigation with public transports is shown in Figure 3.12. The mockup of settings preferences is shown in Figure 3.13. Finally, the mockup of notifications provided by the application is shown in Figure 3.14.



18:00 @ Politecnico - Leonardo

Figure 3.11: Navigation with car mockup.

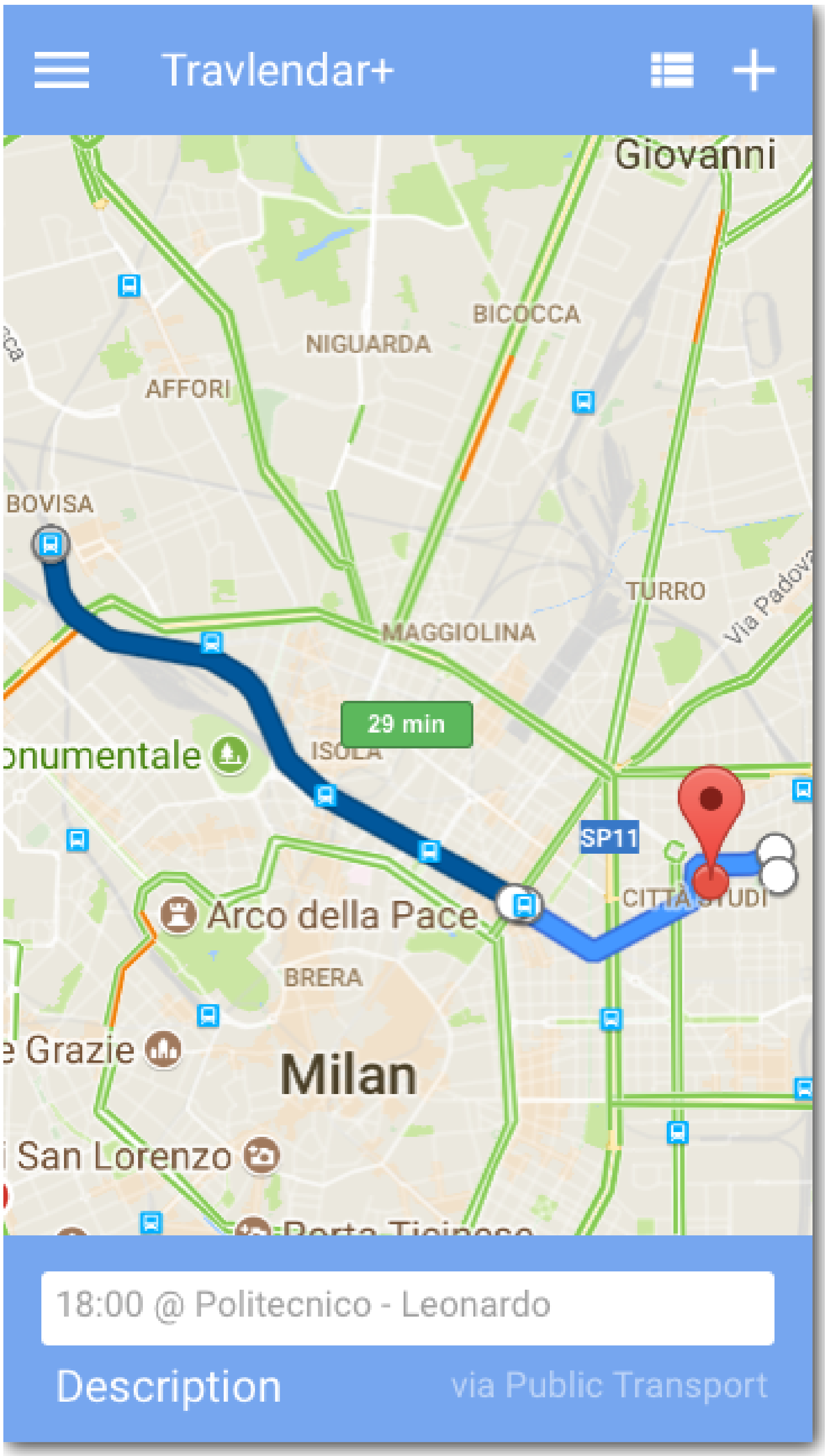
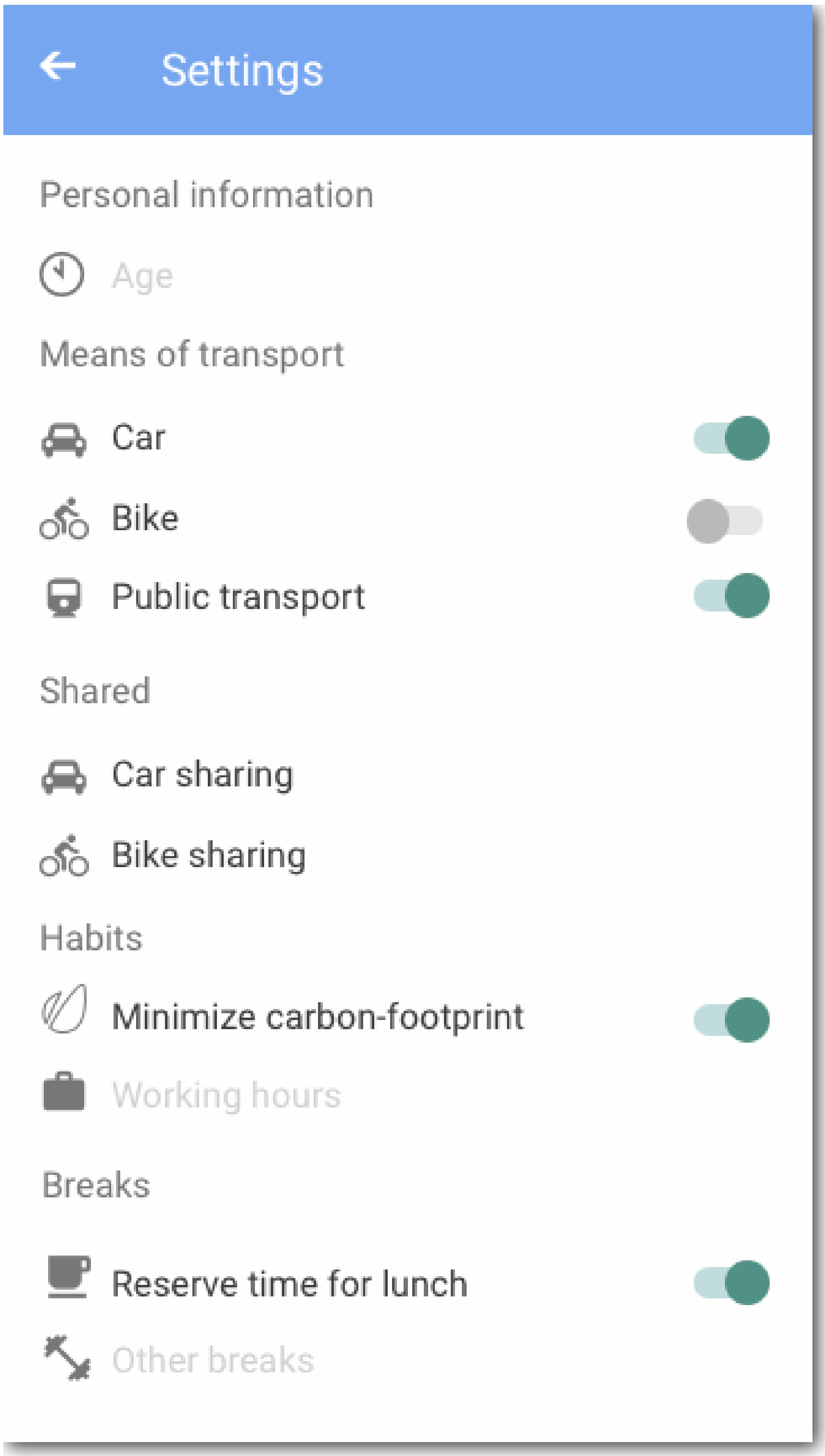


Figure 3.12: Navigation with public transports mockup.



Personal information

Age

Means of transport

Car ●

Bike ○

Public transport ●

Shared

Car sharing

Bike sharing

Habits

Minimize carbon-footprint ●

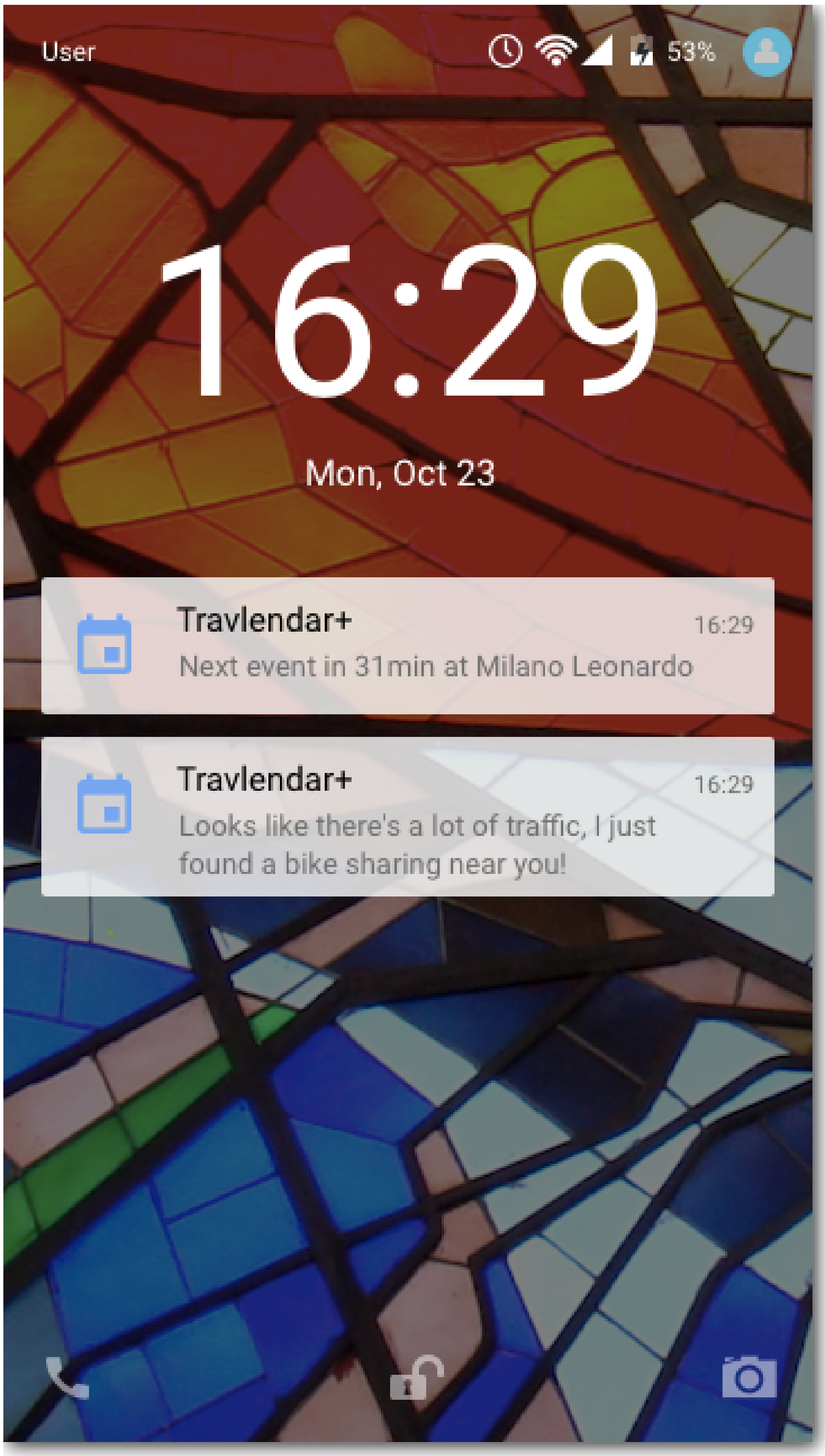
Working hours

Breaks

Reserve time for lunch ●

Other breaks

Figure 3.13: User preferences settings mockup.



Travlendar+ 16:29
Next event in 31min at Milano Leonardo

Travlendar+ 16:29
Looks like there's a lot of traffic, I just found a bike sharing near you!

Figure 3.14: Application notification mockup.

Purchase tickets and use rental service

Purpose

Another great feature of Travlendar+, besides the capability of displaying into the user's events schedule which kind of transport means the user should take to reach the location of each appointments, he/she can take public transportation by buying tickets directly within the application or by inserting his pass code. If a user should take a taxi, he can easily book it by calling the local taxi service company. The application, after locating a shared transport of a vehicle sharing system, such as a bike or car, will allow the user to pay with his credit card from the application.

Scenario 1

Alice is a 22-year-old student and she uses Travlendar+ to organize university courses between campus Bovisa and campus Leonardo. Alice does not use the car and among the system setting she has set public transport as means of transportation. Travlendar+ will suggest Alice to use either public transport (including taxi service) or vehicle sharing services. After entering all the lecture of the day, Travlendar+ advises Alice the meter as means of transport to take between lessons at Campus Bovisa and lessons at Campus Leonardo. Alice, by simply accessing the general information of the event, will be able to buy a metro ticket through the local public transport application. Once she has purchased the ticket, in general information of the event, Travlendar+ shows to Alice the QR code, which will allow her to take the subway in order to make the desired transfer.

Scenario 2

Bob is a personal trainer who lives in Milan, and wants to organize his fitness courses through Travlendar+ application. After registering and setting general preferences for the means of transport to be use, Bob adds a course he will have to do in the afternoon. The application automatically calculates every possible route and shows him the fastest, which must be run with the bicycle. Unfortunately, Bob does not have a bike available at this time, so he does not know how to reach the location. However, Travlendar+ incorporates a feature that allows the user to use local bike sharing services simply by accessing through the general information interface of the event. Bob, by clicking on the bike sharing service (e.g. Mobike), can locate the bikes closest to him and takes the one that is most comfortable in order to reaching the destination as fast as possible.

Use case

The use case diagram for mobility selection is shown in Figure 3.15, whereas the use case diagram for purchasing tickets is shown in Table 3.6.

Sequence diagram

The sequence diagram of the purchase tickets process is illustrated in Figure 3.16, whereas the sequence diagram of the vehicle sharing process is illustrated in Figure 3.17.

Associated functional requirements

1. The system offers the possibility to buy public transportation tickets when a bus, a metro or a tram should be taken.

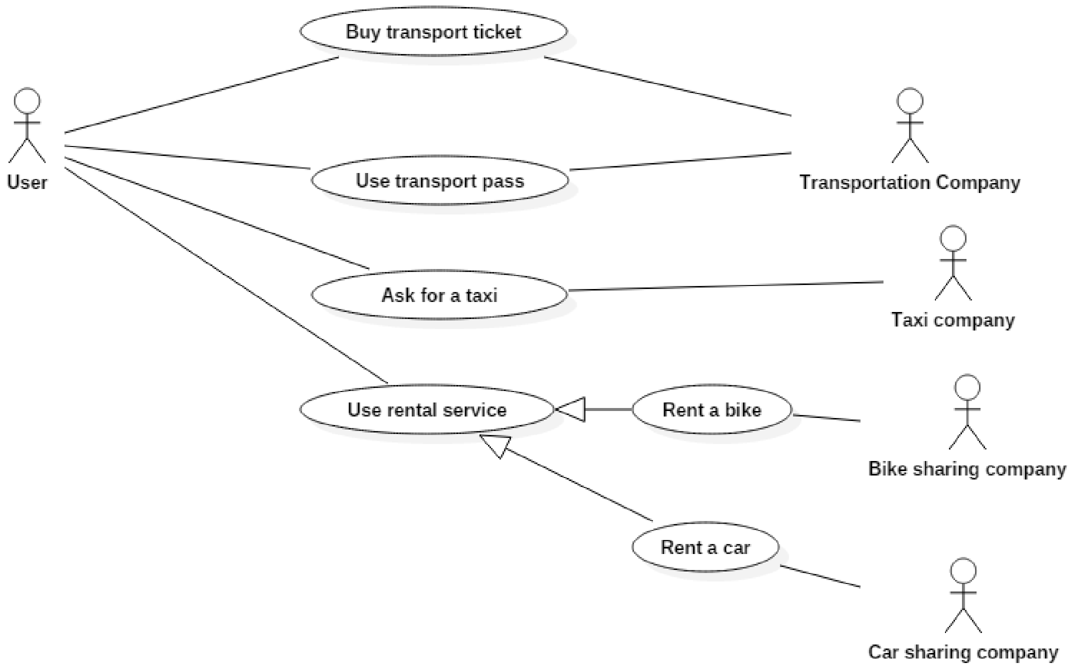


Figure 3.15: Use case diagram for mobility selection

Name	Purchase Tickets
Actors	User
Input condition	The user has already added at least one appointment and has been validated
Flow of events	1. The user may want to buy a public transport ticket 2. The system forward the request to an external service in order to process the data and check the availability of the ticket. 3. The external transport company API checks if there are tickets which are still purchased and asks to user the confirmation of buy the ticket.
Output condition	The QR code of the tickets that he has purchased are added into the specific windows where the user can manage all his public transport ticket.
Exception	The ticket may not be available for a specific means of transport (metro, bus, tram)

Table 3.6: Use case for purchasing tickets

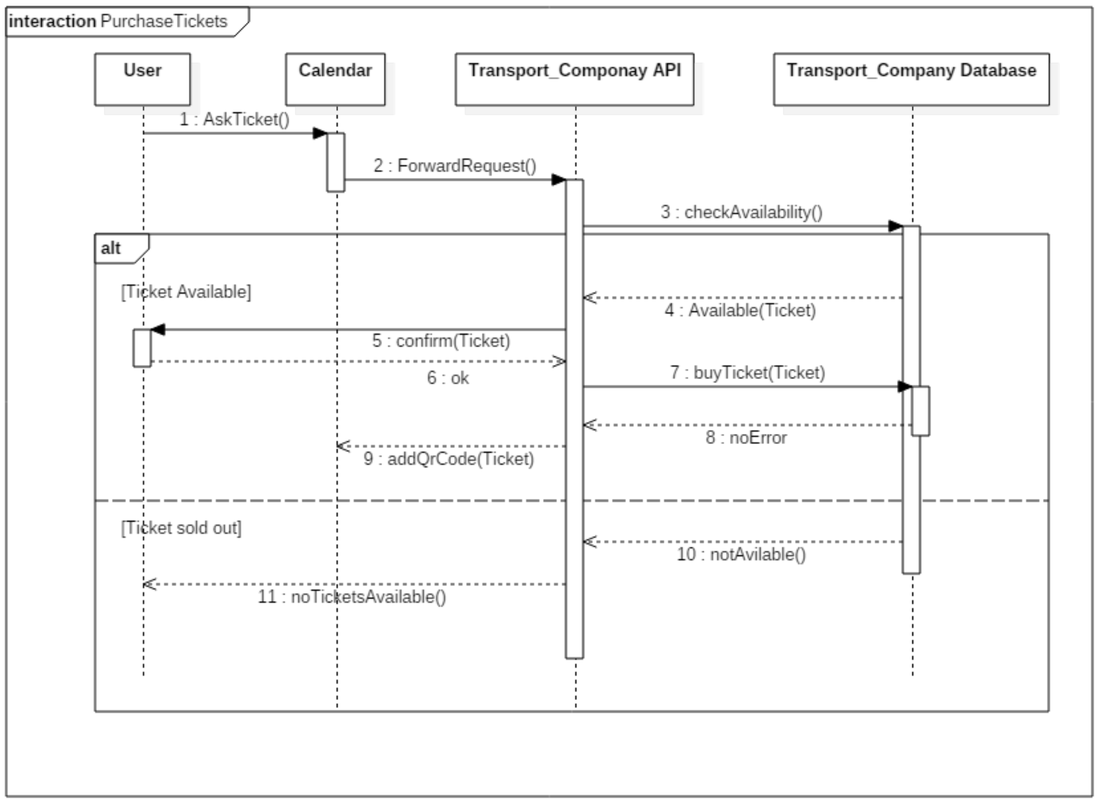


Figure 3.16: Sequence diagram: Purchase Tickets

2. The user can add his personal day/week/season pass when he has to take public transportation.
    - When a user has to take a public transport he can type in his pass code. The system automatically checks the pass code by forwarding the request to the local public transportation API.
3. The system displays all the possible vehicle sharing system in the general information interface of the event, if it should be taken. In particular the two ones are:
    - Car sharing system (e.g. Enjoy, Car2go)
    - Bike sharing system (e.g Mobike)
4. The system allows the user to locate the vehicle available around him, by accessing to the map that the application provides.
    - When the user has to take a car, he can reserve it and access to all the information that he needs through the application.
    - When the user has to take a bike, he can locate it through the map and take it through the specific sharing service API.
5. The system includes a taxi service, that allows user to reserve a taxi by calling the local taxi service with the number that the application shows to him.

Mockups

The mockup of the ticket purchase is shown in Figure 3.18.

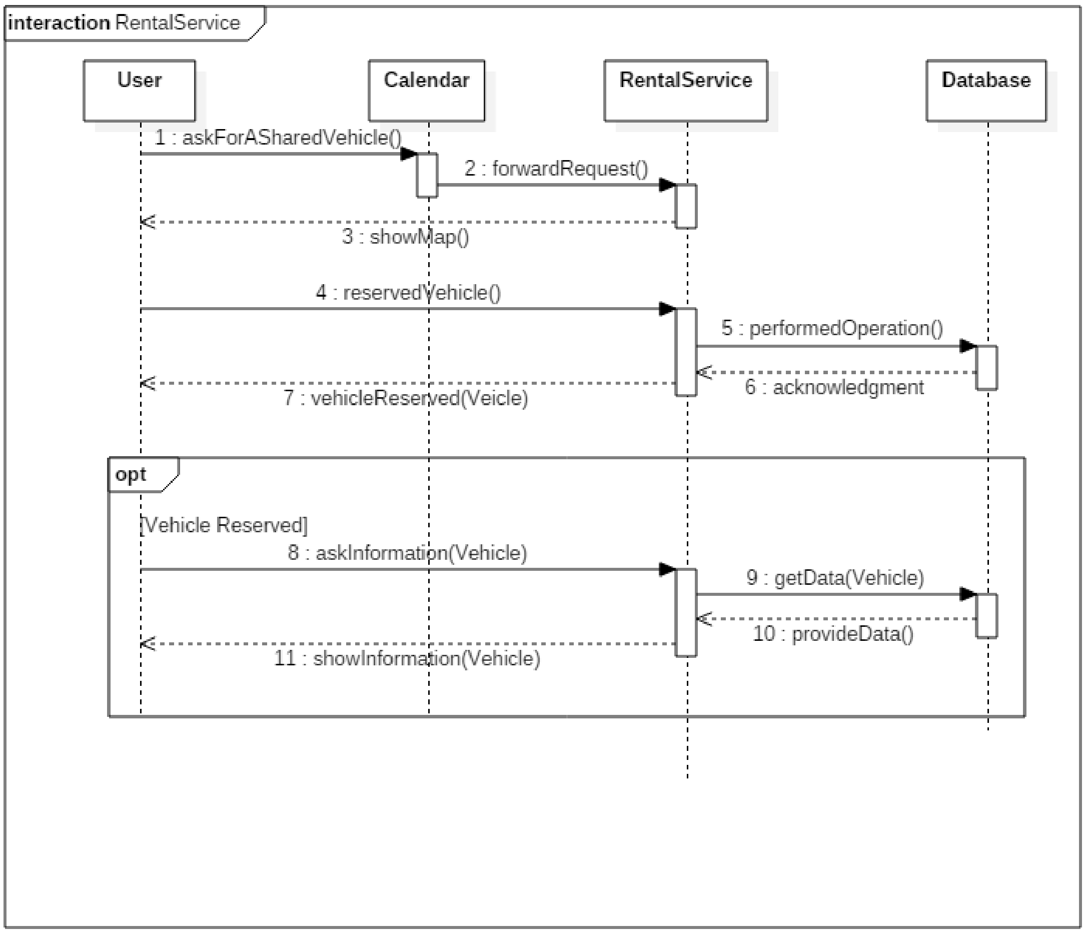
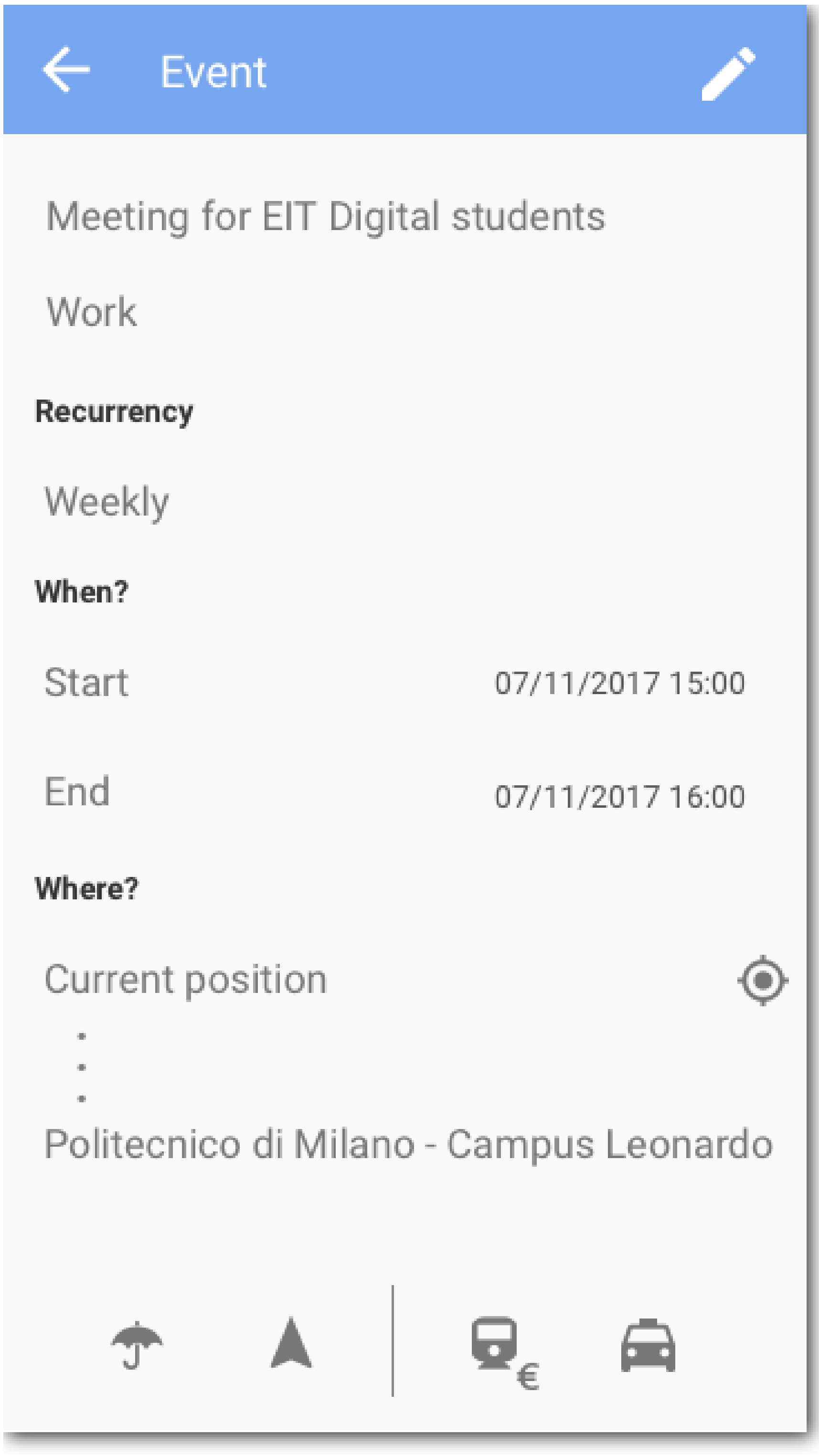


Figure 3.17: Sequence diagram: Vehicle Sharing



Event

Meeting for EIT Digital students

Work

Recurrency

Weekly

When?

Start 07/11/2017 15:00

End 07/11/2017 16:00

Where?

Current position

Politecnico di Milano - Campus Leonardo

Figure 3.18: Purchase ticket mockup.

3.3 Performance Requirements

The user experience across the application should be fluid and with zero waiting time moving between the sections. Besides, there are a number of requirements that implies third party information retrieved through asynchronous requests so, the previous requirement can allow some tolerance.

1. There is no limit to the number of users registered to the application.
2. There is no limit to the number of the simultaneous users of the application.
3. The login process must be completed in less than 5 seconds once submitted the data.
4. The date text fields have to validate dates in real time.
5. The application going back to foreground has to be in the same state as when it was going to background

3.4 Design Constraints

3.4.1 Standards compliance

The application will be developed with the MVVM architectural pattern to allow the unification of the business logic and preserving the coherence across all the various implementations. The implementation on iOS and Android will be compliant with the App Store Review guidelines and Android Compatibility Definition Document. The web-app platform must be compliant with the W3C standards.

3.4.2 Hardware limitations

The application relies on the usage of the GPS location and Internet access. In any case of disabled GPS or energy-saver option that prevents the retrieval of the User's location or in case of no connectivity the application will not provide the user the ETA or the route navigation to reach an event nor the login function if the user is logged out.

3.4.3 Any other constraint

There's no other specific constraint.

3.5 Software System Attributes

3.5.1 Reliability

The application should have a 100% uptime in each designed scenario. Since the application relies on different systems API's there is room to accept small shifts in this requirement.

3.5.2 Availability

Concerning the mobile implementations the application will be available in the App Store and Play Store, rather, the desktop implementation will be available through the web app. Eventually, there could be implementations of the project with Xamarin.Mac and UWP on Mac and Desktop available through the App Store and Microsoft Store.

3.5.3 Security

The applications will perform every request in HTTPS to preserve the security of the user. Data that have to be stored in the device, will be encrypted with third-party libraries to avoid leakage of the user's information as stated in the Software Interface paragraph.

3.5.4 Maintainability

The implementations will be developed with the MVVM architectural pattern allowing a modular integration of future new features or changes in the requirements. The application will rely on Google Maps API in order to maintain its core functionalities and not on other third-party libraries that could be deprecated in future OS versions.

3.5.5 Portability

The clients are implemented with Xamarin giving support to OS's 3 versions older than the last one available for each platform. (iOS 9-10-11, Android 6,7,8) The applications will rely upon the MvvmCross framework to have the ecosystem needed to share the most code across the platforms. The web-app will be based on AngularJS. Our back-end service will be based on AWS Cognito Identity Services to allow scalability and integration with the business logic that is required. Changes from the normal workflow will be addressed with AWS Lambda or EC2 instances whenever needed.

Chapter 4

Alloy Analysis

The following is a possible solution of the Alloy model. We present a completed version of such model in Figure 4.1 and a reduced version with only the main entities, shown in Figure 4.2.

```
open util/boolean

abstract sig EventType { }

lone sig LUNCH extends EventType { }
lone sig WORK extends EventType { }
lone sig HOME extends EventType { }
lone sig BIRTHDAY extends EventType { }
lone sig HOLIDAY extends EventType { }

abstract sig TravelMeanType { }

lone sig CAR extends TravelMeanType { }
lone sig WALKING extends TravelMeanType { }
lone sig BICYCLE extends TravelMeanType { }
lone sig PUBLIC_TRANSPORT extends TravelMeanType { }

abstract sig AccidentType { }

lone sig STRIKE extends AccidentType { }
lone sig BAD_WEATHER extends AccidentType { }
lone sig OUT_OF_SERVICE extends AccidentType { }


sig Float { }
lone sig Stringa { }

sig Date {
        day: one Int,
        time: one Int
}

sig Position {
        latitude: one Int,
        longitude: one Int
}

one sig User{
        userName: one Stringa,
//      name: one Stringa,
//      surname: one Stringa,
        email: one Stringa,
//      phoneNumber: one Int,
        actualPosition: one Position,
        calendar: one Calendar
}

sig Calendar{
        user: one User,
        event: some Event
}

sig Transportation {
        travelMean: one TravelMeanType,
        accidentType: lone AccidentType,
        isShared: one Bool,
        isAccident: one Bool
}{
        (travelMean = PUBLIC_TRANSPORT or travelMean = WALKING) implies (isShared = False)
        (one accidentType) implies (isAccident = True)
}

sig Location {
        location: lone Position,
        transport: lone Transportation,
        isReachable: one Bool
}{
        /* GPS off? */
        (location.latitude =  0 and location.longitude = 0) implies (no
        transport and isReachable = False) else (one transport and isReachable = True)

        (no transport) implies isReachable = False
}

sig Event {
//      eventName: lone String,
//      currentPosition: one Position,
        destination: one Location,
        startDate: one Date,
        endDate: one Date,
        eventType: lone EventType,
        recurrence: lone Int,
        calendar: one Calendar,
        isAllDay: one Bool
}{
        startDate.day ≤ endDate.day

        (eventType = LUNCH) implies (startDate.day = endDate.day and isAllDay = False)

        (isAllDay = True) implies (startDate = endDate)
}

/*            FACTS            */
/* Association 1-1 between Calendar and User */
fact associationCalendarUser {
        all c:Calendar, u:User |
                (c in u.calendar <=> (u in c.user))
}

/* Association 1-1 between Calendar and Event */
fact associationCalendarEvent {
        all c:Calendar, e:Event |
                (c in e.calendar <=> (e in c.event))
}

/* No multiple users with same email or username */
fact userUnivocity {
        no u1, u2: User | (u1 ≠ u2) and (u1.userName = u2.userName and u1.email = u2.email)
}

/* Different calendar */
fact calendarUnivocity {
        no c1,c2: Calendar | (c1 ≠ c2) and (c1.user = c2.user)
}

/* An event that starts and ends in the same day, must has startTime lower than endTime */
fact {
        all e:Event | (e.startDate.day = e.endDate.day and e.isAllDay = False)
                implies
                e.startDate.time < e.endDate.time
}

/* An event that starts and ends in the different days, can have any startTime or endTime */
fact {
        all e:Event |e.startDate.day ≠ e.endDate.day
                implies (one e.startDate.time and one e.endDate.time)
}

/* Two events must not overlap */
fact noEventAtSameTime {
        all e1,e2: Event | (e1 ≠ e2) and (e1.startDate.day + e1.endDate.day = e2.startDate.day
            ↪  + e2.endDate.day)
                implies
                (e1.endDate.time < e2.startDate.time or e2.endDate.time < e1.startDate.time)
}

/* There must be one event corresponding to lunch per day */
fact oneLunchPerDay {
        no e1, e2: Event | e1 ≠ e2 and e1.eventType = LUNCH and e2.eventType = LUNCH and e1.
            ↪ startDate.day + e1.endDate.day= e2.startDate.day + e2.endDate.day
}

/* If there's an accident of kind bad weather, travel means like walking and bicycle cannot be
    ↪  used */
fact {
        all t: Transportation | t.accidentType = BAD_WEATHER implies (t.travelMean ≠ WALKING
            ↪ and t.travelMean ≠ BICYCLE)
}

/* If there's an accident of kind strike or out of service, public transportation cannot be
    ↪ used */
fact {
        all t: Transportation | t.accidentType = STRIKE or t.accidentType = OUT_OF_SERVICE
            ↪ implies (t.travelMean ≠ PUBLIC_TRANSPORT)
}

/* Every Location must have at least one event associated */
fact {
        all l: Location | some e: Event | l in e.destination
}

/* Every location that does not have transport must be unreachable */
fact {
        all l: Location | #l.transport = 0
                implies
                l.isReachable = False
}

/*                      ASSERTION                          */
assert noEventOverlapping {
        all e1,e2: Event | (e1 ≠ e2) and (e1.startDate.day + e1.endDate.day = e2.startDate.day
            ↪  + e2.endDate.day)
                implies
                (e1.endDate.time < e2.startDate.time or e2.endDate.time < e1.startDate.time)
}

//check noEventOverlapping
//OK

assert noEventWithoutCaldendar {
        all e: Event | #e.calendar = 1
}

//check noEventWithoutCaldendar
//OK

assert allDayEvents {
        no e:Event | e.isAllDay = True and ( e.startDate ≠ e.endDate)
}

//check allDayEvents
//OK

assert noEventWithEndDateLowerThanStartDate {
        no e: Event | e.startDate.day = e.endDate.day and e.isAllDay = False and (e.startDate.
            ↪ time > e.endDate.time)
}

//check noEventWithEndDateLowerThanStartDate
//OK

assert noEventWithEndDateBeforeStartDate {
        no e: Event | e.startDate.day < e.startDate.day
}

//check noEventWithEndDateBeforeStartDate
```

```
//OK

assert noLocationUnreachableWithTransportAssociated {
        no l: Location | l.isReachable = False and #l.transport =  1
}

//check noLocationUnreachableWithTransportAssociated
//OK


/*      PREDICATES                      */

pred show() {
        #Location ≥ 1
        #Event ≥ 1
        #Position ≥ 1
        #Transportation ≥ 1
        #{t: Transportation | t.isShared = True} = 1
        #{e: Event | e.eventType = LUNCH} = 3
}

run show for 10
```

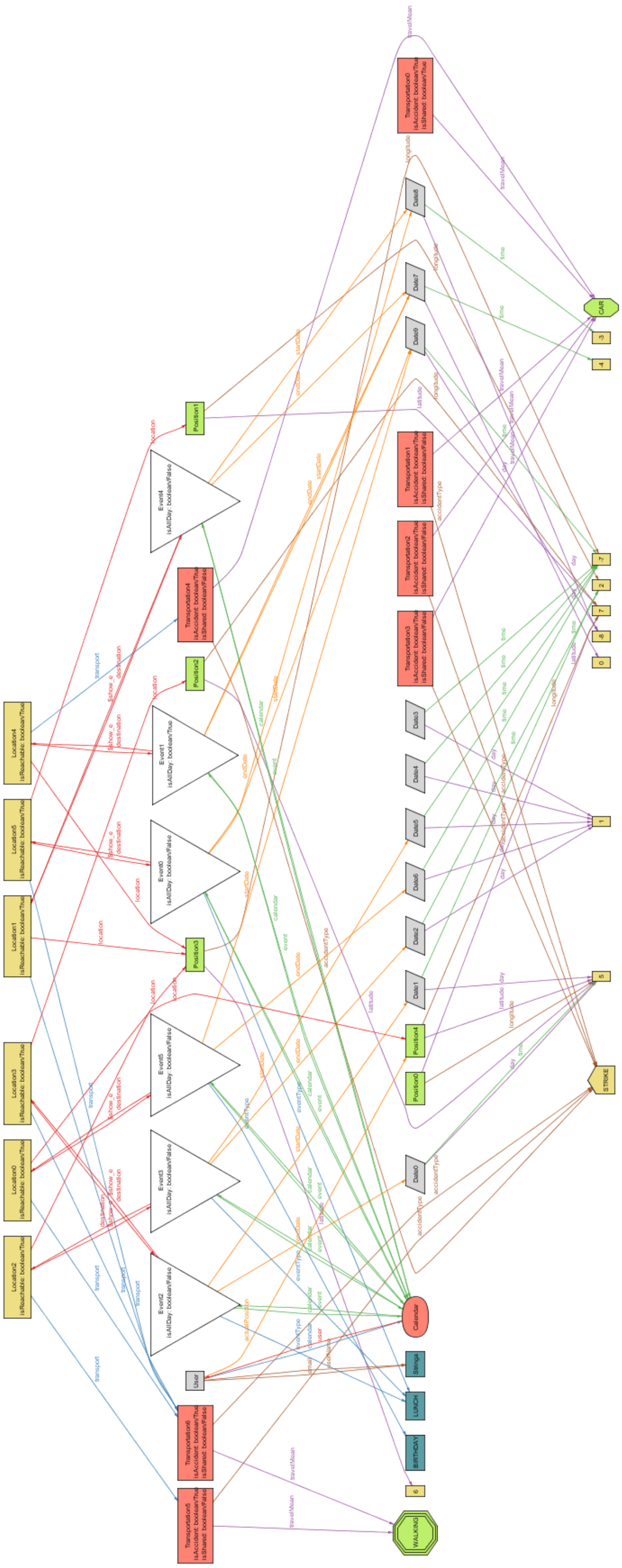


Figure 4.1: Alloy model (complete version).

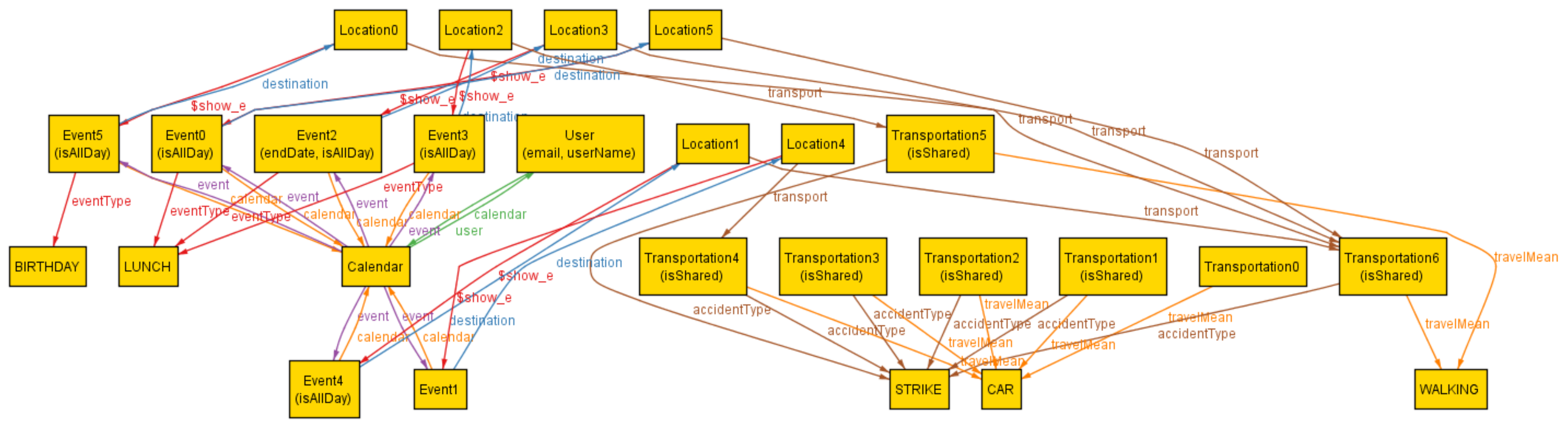


Figure 4.2: Alloy model (reduced version).

Bibliography

[1] ISO/IEC/IEEE standard 29148:2011: International Standard - Systems and software engineering - Life cycle processes - Requirements engineering. http://ieeexplore.ieee.org/document/6146379/. Last time retrieved: 25th of October, 2017.

[2] IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications. http://ieeexplore.ieee.org/document/392555/. Last time retrieved: 25th of October, 2017.

[3] Elisabetta Di Nitto and Matteo Rossi. *AA 2017-2018 Software Engineering 2 — Mandatory Project goal, schedule, and rules*. 2017.

[4] MIT - Alloy: a language & tool for relational models. http://alloy.mit.edu/alloy/. Last time retrieved: 25th of October, 2017.

Appendix

Software and tools used

- Overleaf.com[1] for writing in LaTeX this document, a real-time collaborative web application.
- Slack.com[2] for communicating with each other, sharing ideas, and collaborating.
- GitHub[3] for revision control.
- UXpin.com[4] for prototyping and drawing mockups.
- StarUML[5] for creating UML diagrams.
- Alloy Analyzer[6] for modeling the system.

Hours of work

All statistics about commits and code contribution are available on our repository on GitHub. Please, remind that all commits on our repository have been reviewed together. Also the whole work has been equally divided and everyone helped each other.

- Antonio Frighetto: 29 hours.
- Leonardo Givoli: 28 hours.
- Hichame Yessou: 28 hours.

[1] https://www.overleaf.com
[2] https://slack.com/
[3] https://github.com
[4] https://www.uxpin.com
[5] http://staruml.io/
[6] http://alloy.mit.edu/alloy/