Software Engineering 2

Travlendar+

Requirements **A**nalysis and **S**pecification **D**ocument

Menchetti Guglielmo
Norcini Lorenzo
Scarlatti Tommaso

October 29, 2017

Contents

1 Introduction

The following Requirement Analysis and Specification Documents aims at providing a baseline for project planning and development. It illustrates the specifics of the application domain and the relative System in terms of functional requirements, non functional requirements and constraints. It describes the components and the System interactions with the external world, also through use cases scenarios. A more formal specification of the most relevant features will be specified with the use of the Alloy language.
This document is intended for developers tasked with the implementation of the System, testers who are involved in requirements validation and project managers that supervise the development process.

1.1 Revision History

Version	Date	Authors	Summary
1.0	29/10/2017	G. Menchetti L. Norcini T. Scarlatti	First release
1.1	27/11/2017	G. Menchetti L. Norcini T. Scarlatti	Main changes • Global changes after Design phase • Changes in software attributes • Changes in goals and requirements section • Changes in the class diagram

1.2 Purpose

Travlendar+ is a calendar based application whose goal is to help registered Users organize their day by scheduling their appointments and providing the best solutions in terms of mobility.
The application allows Users to create a meeting for a specified time, producing a feedback of feasibility according to the other appointments and providing Users with a list of travel options leveraging Users preferences and additional informations (e.g. traffic). Once created, events can be modified

or deleted at any time up to the moment they start. Furthermore, each appointment can be assigned to a specific class (e.g. business, family, personal, etc) previously defined by Users. The System will produce a warning if a new appointment creates any schedule conflicts, for instance if the time overlaps with another meeting or if the location is not reachable on time. In this case, Users won't be allowed to create the event.
Travlendar+ implements a notification System: Users may chose whether to be notified a certain time before the event or not.
Users are able to express their preferences and constraints concerning certain means of travel. For example, traveling by bicycle or on foot is allowed only on distances shorter than a given threshold, public transport or taxis, if available, may be preferred to the use of a personal car.
Other than the standard ones, *Travlendar+* offers two particular type of events: recurrent and flexible. Flexible events can be scheduled in a determined window of time and with a specified minimum duration (e.g., going to gym may imply at least 1.30h of training but does not necessarily need to start at a specific time). Recurrent events instead, can be scheduled for more than one day with a frequency specified by the User.
Finally, *Travlendar+* provides a booking functionality allowing Users to purchase public transports tickets, reserve a taxi or locate the nearest car or bike of a vehicle sharing service through the use of third party platforms.

The above description can be summarized as follows, in a list of goals:

- **[G.1]** The System allows the User to access the functionalities of the application from different locations and devices. The data has to be coherent across different devices.
- **[G.2]** The System allows the User to manage meetings in his schedule.
- **[G.3]** The System allows the User to reach every meeting on time.
- **[G.4]** The System allows the User to select or edit a travel mean to reach an event.
- **[G.5]** The System allows the User to set preferences.
- **[G.6]** The System allows the User to create flexible events and repetitive events.
- **[G.7]** The System allows the User to book transportation for a trip.
- **[G.8]** The System allows the User to be notified before the occurrence of an event.

1.3 Scope

According to the *World and Machine* paradigm we can distinguish the *Machine*, which is System to be developed, from the *World*, which is portion of the real-world affected by the machine. This separation leads to a classification of the phenomena in three different types, depending on where they occur.
In this context, the main relevant phenomena are organized as follows:

World phenomena:

- Transport delay: unforeseen circumstances that cause lateness with respect to the computed ETA.
- Unscheduled appointment: appointments the User either forgot or did not know about in advance.
- Unexpected issues: anything that leads to a delay or even a cancellation of the meetings.

Machine phenomena:

- API queries: requests for third-party services.
- Feasibility study: the System checks the schedulability of a new or edited event.
- System clock: the System runs its tasks according to its clock.
- DBMS: all operations performed to retrieve/store data.

Shared Phenomena:

Controlled by the world and observed by the machine

- Registration/Login: a guest can sign up to the application or otherwise log in if is already registered.
- Manage events: the User can create, modify or delete events.
- Set preferences: the User customizes his/her travel experience.

Controlled by the machine and observed by the world

- Suggest travel options: provides a User with a list of possible travel means.
- Generate warnings: produce a warning in case of a scheduling conflict.
- Send notifications: alert the User before an event.

1.4 Definitions, Acronyms, Abbreviations

1.4.1 Definitions

- *Guest:* a person who has to sign up and who is not able to access any feature of the application.
- *User:* a registered User who has to log in.
- *Logged in User:* a registered User who logged in.
- *Calendar:* User personal calendar that shows the events organized per day, week or month.
- *Schedule:* the set of the events of a User for a specific day.
- *Event, Meeting, Appointment:* different ways to refer to entries in the schedule.
- *Flexible Event:* a particular type of event that can be scheduled in a time slot and with a given duration.
- *Repetitive Event:* an event that is scheduled more times with a specified frequency.
- *Warning:* a message sent from the System to the User to warn him/her about a conflict in the schedule.
- *Notification:* a message which advise the User of an upcoming event.
- *Third-party services:* services used by the System in order to provide extra functionalities.

1.4.2 Acronyms

DBMS	Data Base Management System
HTTPS	Hyper Text Transfer Protocol Secure
ETA	Estimated Time of Arrival
API	Application Program Interface
IEEE	Institute of Electrical and Electronics Engineers
OS	Operative System
UI	User Inteface

1.4.3 Abbreviations

- [**Gn**]: n-th goal
- [**Dn**]: n-th domain assumption
- [**Rn**]: n-th functional requirement

1.5 Reference Documents

- IEEE Standard 830-1998: Recommended Practice for Software Requirements Specifications
- IEEE Standard 830-1993: IEEE Guide to Software Requirements Specifications
- Alloy documentation
- Project assignment specifications

1.6 Overview

The rest of the document is organized in this way:

- **Overall description:** in this section a general description of the application will be given, focusing on the context of the System and giving further details about shared phenomena. The most relevant functions of the System are also explained, together with the User characteristics. Finally, through the specifications of constraints, dependencies and assumptions, we will show how the System is integrated in the real world.
- **Specific requirements:** in this section will be provided a full-detailed perspective of all the aspects of the previous section. The main aim is to give a useful item for both designers and testers.
  Requirements are fully explained and organized according to their type, and a list of all possible interactions with the System is provided through use cases and sequence diagrams.

2 Overall Description

2.1 Product Perspective

Here we discuss in details all the shared phenomena outlined in the previous section, and we provide an oversight of the domain model in different levels of specification, by means of class and state diagrams.

Registration/Login (world controlled, machine observed)
The User can submit his/her credentials in order to register or log in if is already registered. The application provides a minimal and intuitive form which, once filled and submitted, collects, checks and then sends all the data to the associated DBMS.

Manage events (world controlled, machine observed)
The User is able through the calendar to add, edit or delete events from the schedule. In the first two cases, the System will check if the information inserted in the fields are consistent and complete and will send them to the DBMS.

Set travel preferences (world controlled, machine observed)
The User is able to customize his/her travel means preferences in the homonym section provided by the System. Here, every mean has a specific tab with a list of applicable constraints. The User can also globally activate/deactivate a particular travel mean. The application will store all the preferences selected by the User and will use them to suggest the best travel options.

Suggest travel options (machine controlled, world observed)
When the User adds or edits an event, the System shows up all the possible traveling options computed according to the specified preferences. The User can then select one of them.

Generate warnings (machine controlled, world observed)
When a User tries to add a new event to the daily schedule, the System computes a study of feasibility to check if it creates any schedule conflict. If it does, the System will display a warning message to advise the User that the new event cannot deal with the previous ones and the event won't be added to the schedule.

Send notifications (machine controlled, world observed)
If the notification option is enabled, the System will advise the User through

an alert message. The System computes the instant to send the notification according to its internal clock.

2.1.1 Class Diagram

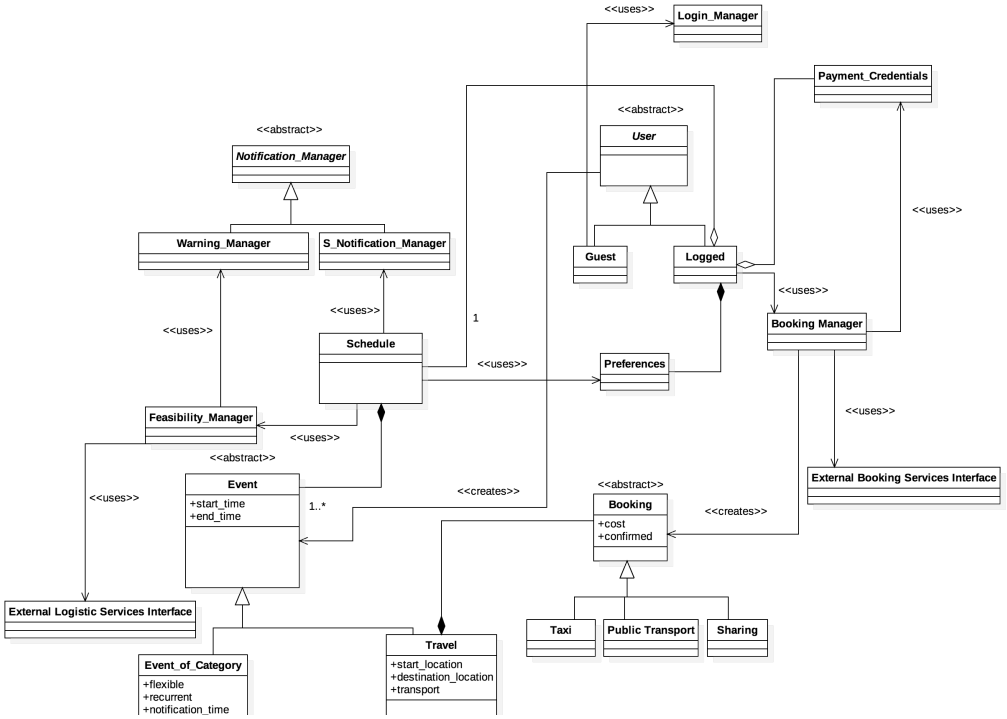


Figure 1: *Travlendar+* Class Diagram

2.1.2 State Chart Diagrams

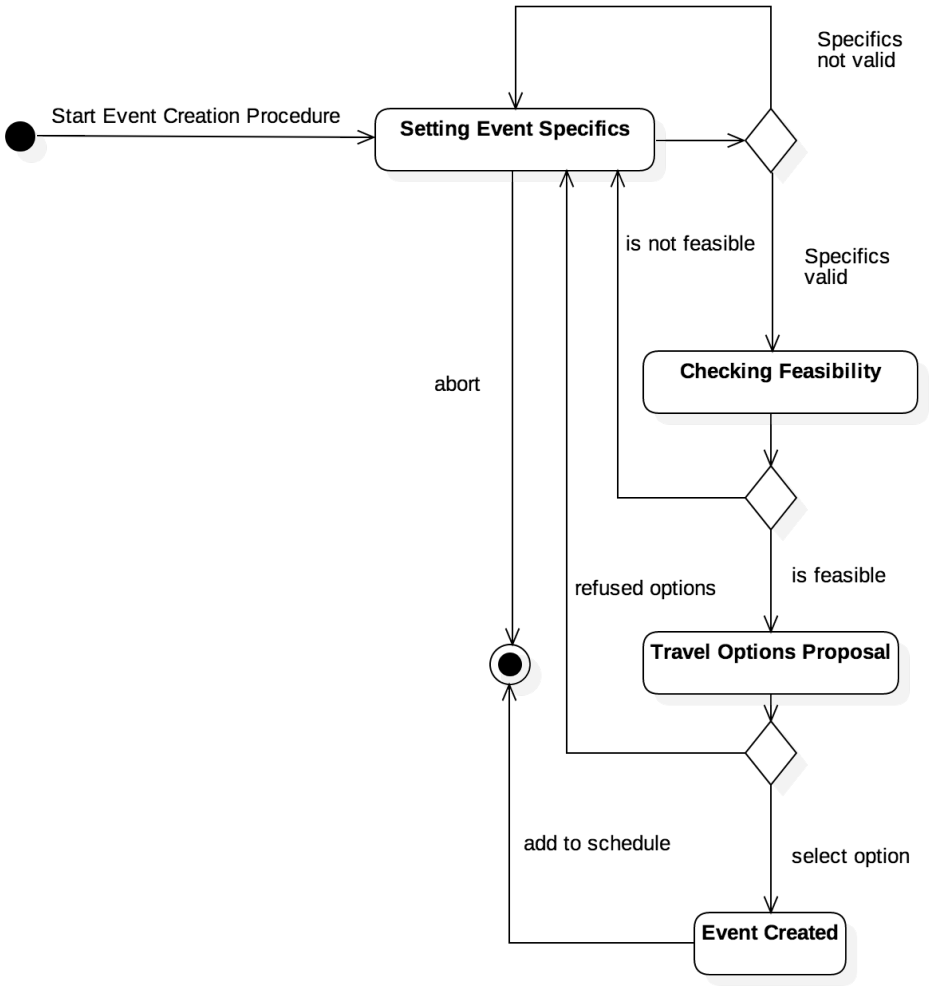


Figure 2: Add event state chart

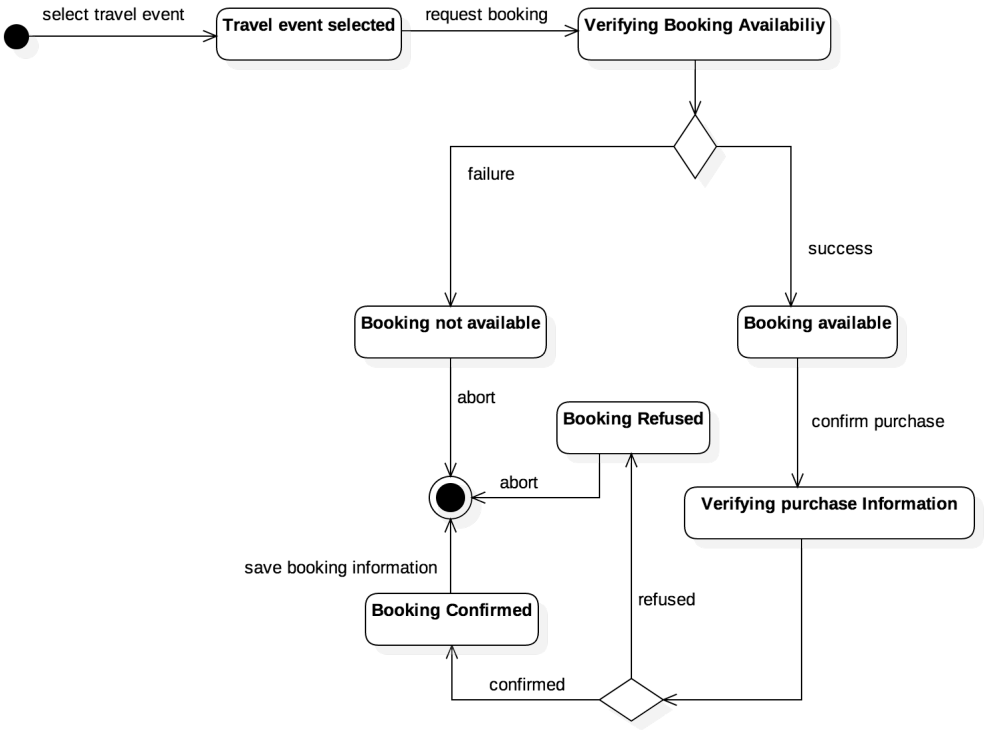


Figure 3: Booking travel state chart

2.1.3 Use Case Diagrams

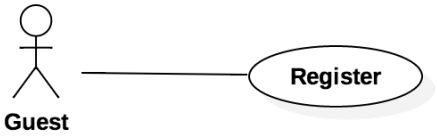


Figure 4: Guest use case

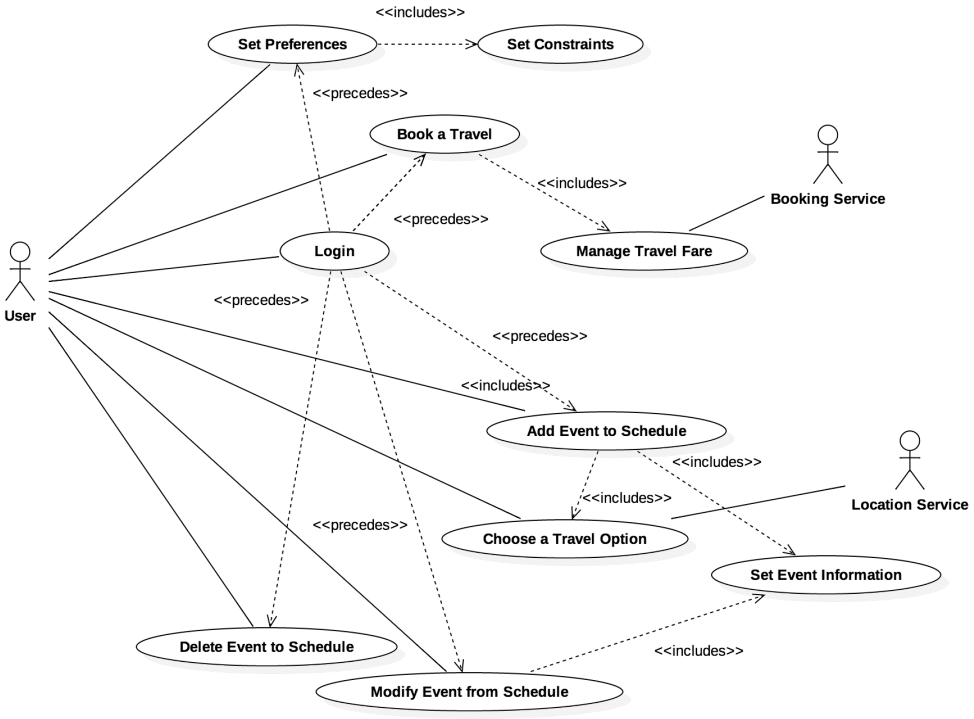


Figure 5: User use case

2.1.4 Use Case Templates

Register Account

ID	UC1
Description	The ***Guest*** wants to create an account for the application.
Actors	***Guest***
Preconditions	The ***Guest*** opened the application
Flow of Events	1. The ***Guest*** selects the function *Sign Up*. 2. The ***System*** returns a form to enter all the required data: username, email address and password which will be used for the future logins. 3. The ***Guest*** fills the form with all the required information. 4. The ***User Manager*** checks if all the informations are correct, generates a random activation URL and asks the ***Mailing System*** to forward his/her URL to the email address of the ***Guest***. 5. The ***Guest*** receives the mail and click on the URL. The ***User Manager*** stores the data provided by the ***User***.
Postconditions	The ***Guest*** is able to sign in.
Exceptions	1. The ***User Manager*** recognizes invalid information than shows an error message. The flow restarts from point 2.

Log In

ID	UC2
Description	The ***User*** wants to log in to the application.
Actors	***User***
Preconditions	The ***User*** opened the application
Flow of Events	1. The ***User*** inserts his/her credentials. 2. The ***User*** taps the *Sign In* button. 3. The ***User Manager*** checks provided credentials.
Postconditions	The ***User*** is logged in.
Exceptions	1. The ***User Manager*** recognizes invalid credentials then shows an error message. The flow restarts from point 2.

Set User Settings

ID	UC4
Description	The ***User*** wants to set User settings.
Actors	***User***
Preconditions	The ***User*** is logged in to the application.
Flow of Events	1. The ***User*** opens the menu. 2. The ***User*** taps the *User Settings* section. 3. The ***System*** displays the *User Settings* view. 4. The ***User*** changes his/her personal informations and clicks the *Confirm* button.
Postconditions	The ***User*** changed his personal informations.
Exceptions	1. The ***User Manager*** recognizes invalid credentials then shows an error message. The flow restarts from point 3.

Set Travel Preferences

ID	UC5
Description	The ***User*** wants to set travel preferences.
Actors	***User***
Preconditions	The ***User*** is logged in to the application.
Flow of Events	1. The ***User*** opens the menu. 2. The ***User*** taps the *Travel Preferences* section. 3. The ***System*** displays the *Travel Preferences* view. 4. The ***User*** selects/deselects each travel mean and set specific constraint for the selected ones. 5. The ***User*** clicks on the *Confirm* button.
Postconditions	The ***User*** changed his/her travel preferences.
Exceptions	

Add normal and flexible event

ID	UC6
Description	The ***User*** wants to add a normal or flexible event to the schedule.
Actors	***User***
Preconditions	The ***User*** is logged in to the application and is in the *Schedule* or *Calendar* section.

Flow of Events	1. The ***User*** clicks the *Add* button. 2. The ***System*** returns a form to enter all the required data: name, position of the event, previous position (from where the appointment will be reached), start and end time. 3. The ***User*** fills all the required fields. 4. The ***User*** clicks the *Confirm* button. 5. The ***Feasibility Manager*** checks the schedulability of the event. 6. The ***System*** calculates the best travel means options and proposes them to the ***User***. 7. The ***User*** selects a travel mean.
Postconditions	The ***User*** added the event to his/her schedule.
Exceptions	1. The ***Feasibility Manager*** detects a conflict in the schedule. The ***Warning Manager*** displays a warning message. The flow restarts from point 2. 2. The ***User*** did not correctly compile the form before clicking *Confirm*. The flow restarts from point 2.

Add repetitive event

ID	UC7
Description	The ***User*** wants to add a repetitive event to the schedule.
Actors	***User***
Preconditions	The ***User*** is logged in to the application and is in the *Schedule* or *Calendar* section.

Flow of Events	1. The ***User*** clicks the *Add* button. 2. The ***System*** returns a form to enter all the required data. 3. The ***User*** selects the *Repetitive* option. 4. The ***System*** shows the form relative to the repetitive event. 5. The ***User*** fills the form and clicks the *Confirm* button. 6. The ***Feasibility Manager*** checks the schedulability for all the repetitions of the event.
Alternative Flow	7. The ***Feasibility Manager*** detects a conflict in the schedule. 8. The ***System*** asks the ***User*** whether schedule the event only in the days without conflict. 9. The ***User*** chooses to proceed with the creation.
Postconditions	The ***User*** added the repetitive event to his/her schedule.
Exceptions	1. The ***User*** did not correctly compile the form before clicking *Confirm*. The flow restarts from point 2. 2. The ***User*** chooses to stop the creation of the event. The flow restarts from point 4.

Delete event

ID	UC8
Description	The ***User*** wants to delete an event from a schedule.
Actors	***User***

Preconditions	The ***User*** is in the *Home* or in the *Schedule* section of a particular day.
Flow of Events	1. The ***User*** clicks the *Delete* button on a particular scheduled event. 2. The ***Warning Manager*** shows a warning message and ask the ***User*** to confirm his/her choice. 3. The ***User*** clicks the *Yes* button.
Postconditions	The ***User*** deleted an event from the schedule.
Exceptions	1. The ***User*** clicked the *No* button in the warning message.

Edit event

ID	UC9
Description	The ***User*** wants to modify an event from a schedule.
Actors	***User***
Preconditions	The ***User*** is in the *Home* or in the *Schedule* section of a particular day.
Flow of Events	1. The ***User*** clicks the *Edit* button on a particular scheduled event. 2. The ***System*** shows the edit page. 3. The ***User*** can modify the event informations. 4. The ***User*** clicks the *Confirm* button. 5. The ***Feasibility Manager*** checks the schedulability of the event. 6. The ***System*** calculates the best travel mean options and propose them to the ***User***. 7. The ***User*** selects a travel mean.

Postconditions	The ***User*** edited the event.
Exceptions	1. The ***Feasibility Manager*** detects a conflict in the schedule. The ***Warning Manager*** displays a warning message. The flow restarts from point 2.

Book travel

ID	UC10
Description	The ***User*** wants to buy a ticket or book a ride for a travel.
Actors	***User***
Preconditions	The ***User*** is in the *Home* or in the *Schedule* section of a particular day.
Flow of Events	1. The ***User*** clicks the *Buy* button on a specific scheduled event. 2. The ***System*** checks the availability of the requested service and redirect the ***User*** to the specific third party service which will manage the *User's* request. 3. The ***User*** provides his/her credentials to the third party service.
Postconditions	The ***User*** bought a ticket or booked a ride for a travel to reach the associated event.
Exceptions	1. The booking service in point 2 is not available. The ***Warning Manager*** shows a warning message and the booking process stops. 2. The credential provided by the ***User*** in point 3 are not correct. The ***Warning Manager*** shows a warning message and the flow restart from point 3.

2.1.5 Sequence Diagrams

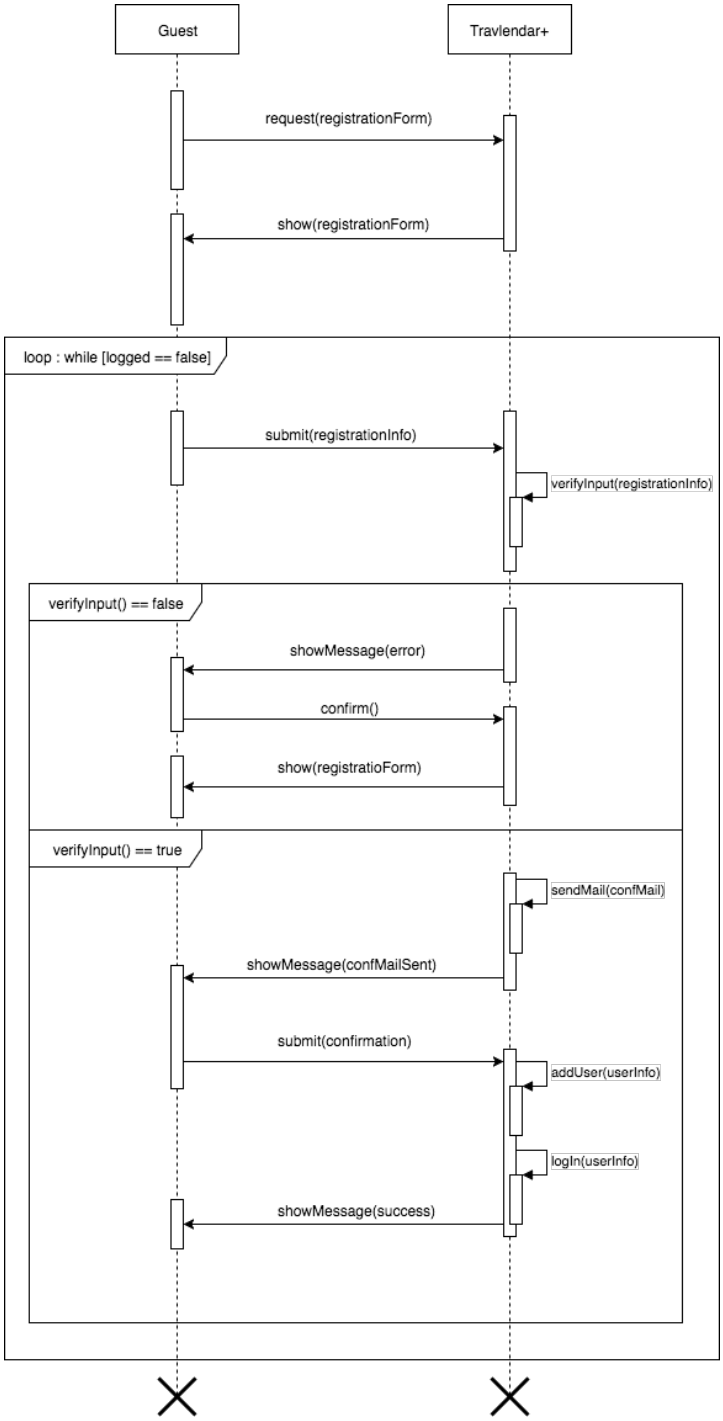


Figure 6: Sign Up sequence diagram

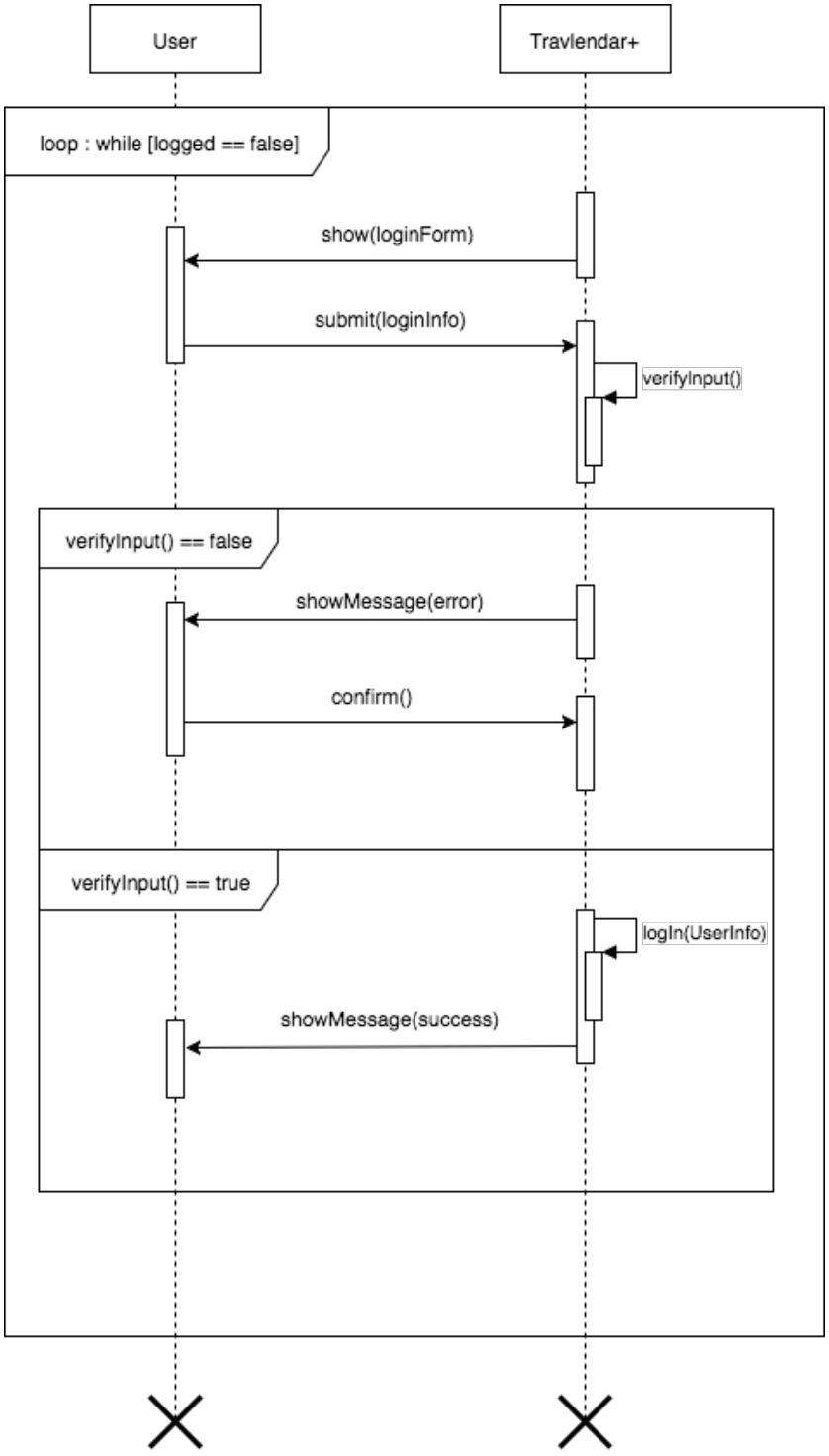


Figure 7: Log In sequence diagram

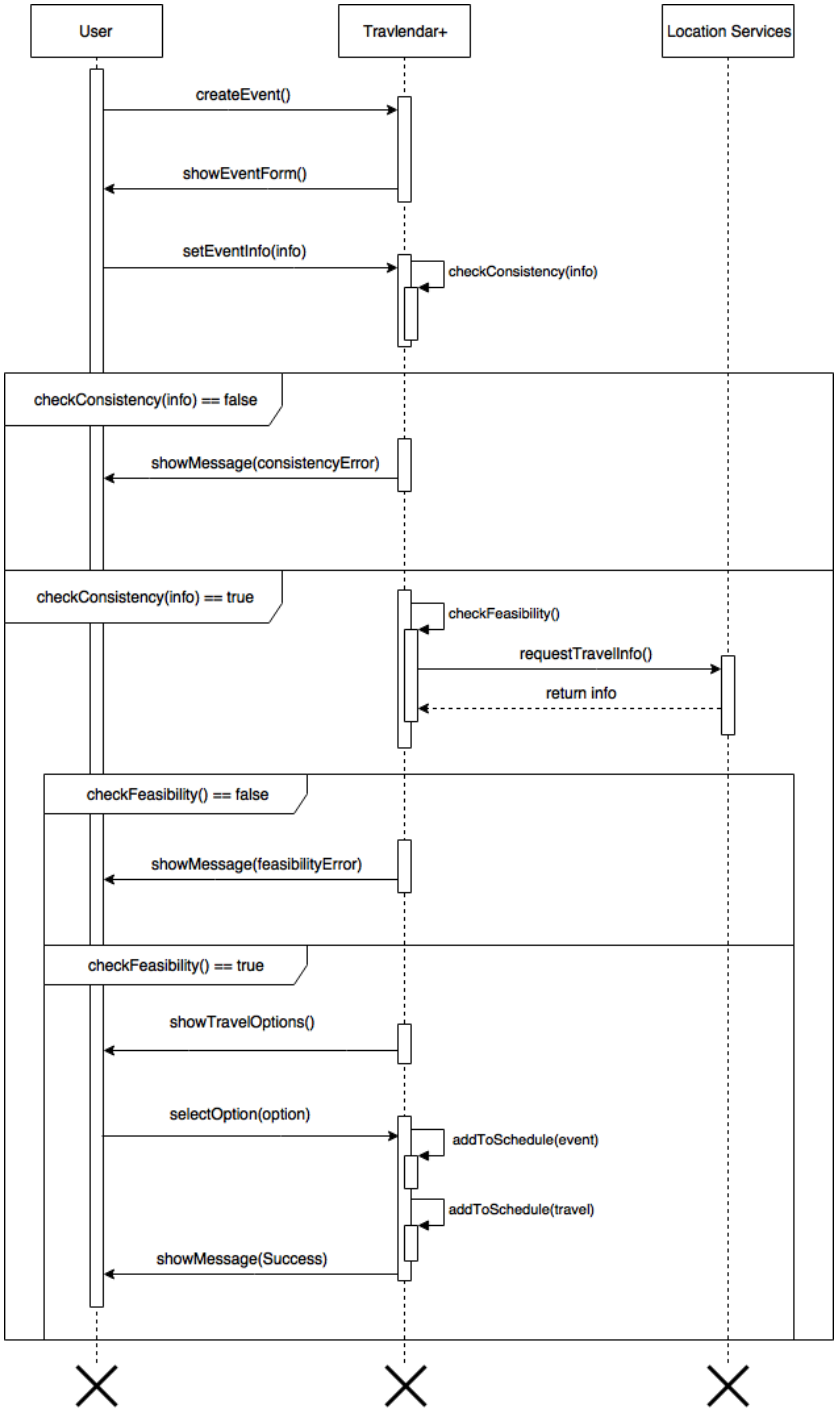


Figure 8: Normal and Flexible event creation sequence diagram

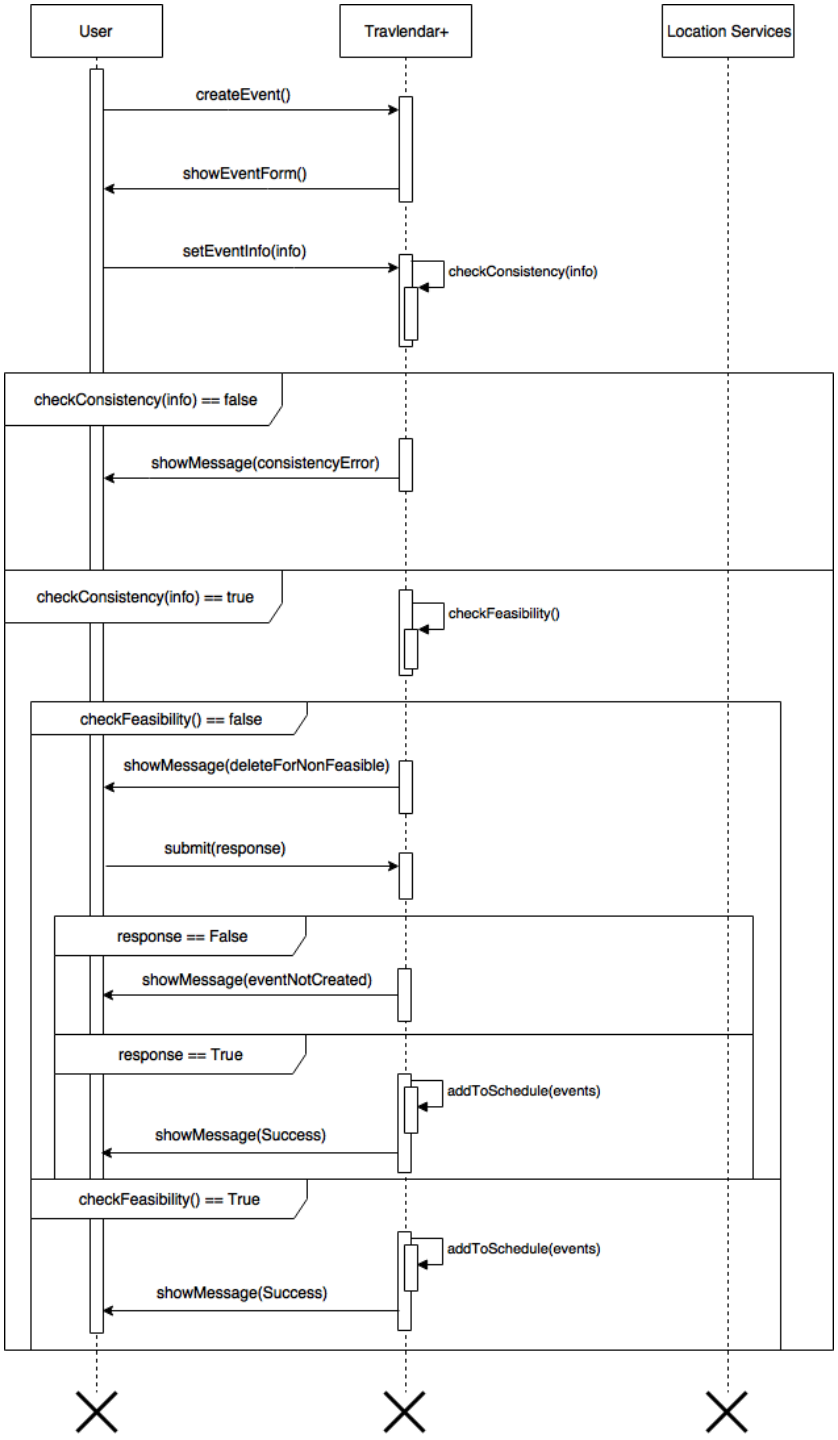


Figure 9: Recurrent event creation sequence diagram

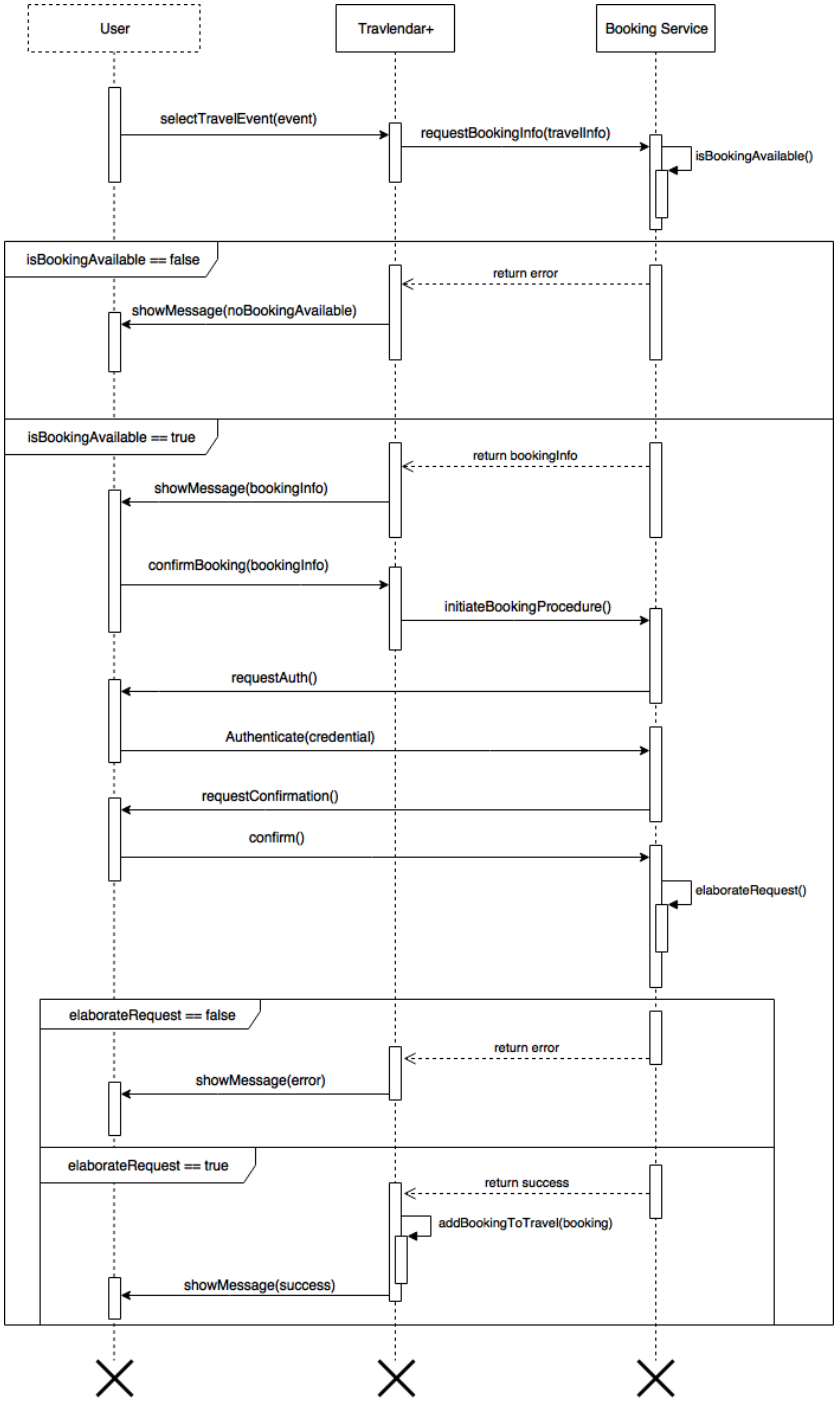


Figure 10: Travel booking sequence diagram

2.2 Product Functions

In this section the most important features of the application are explained.

- **Manage meetings**
  The System allows the User to add, edit and delete events in the schedule for a particular day. During the creation of the event, the User must specify the name of the event, the time and date of start and end of the event, the location of the event and the location from which it will be reached. Furthermore, the User will be able to add additional information about the event such as a briefly description and a category. For each event, the User is also provided with a list of possible travel means: the System will compute the best travel mean options accordingly with the information provided and the Users' travel preferences. The User will then be able to select one of the mean of transportation proposed.
  The User is also able to modify informations about a specific event or to delete it.
  The System guarantees the feasibility of the schedule for each day. To do this, it allows the User to add or edit only events in the schedule that are not in conflict with the ones already present.
  Finally, the User can choose to be notified a certain time before the event.

- **Set travel preferences**
  The User is able to customize his/her travel preferences globally activating or deactivating travel means that are:

  - Personal motor vehicle
  - Public transport
  - Personal bicycle
  - Taxi services
  - Car sharing services
  - Bike sharing services
  - On foot

  Furthermore, for each travel mean a set of editable constraints is provided. The System gives also the possibility to activate an *Eco Mode* option: in this case, the best travel solution will be computed in order to minimize carbon footprint.

- **Create flexible and repetitive events**
  The User is allowed to create flexible and repetitive events. Flexible events are particular type of events in which the User, besides the information about event date and location, must specify a window of time where the event can occur and a duration. In order to preserve the feasibility of the schedule, the System can arbitrarily move the event inside the window's bounds.
  On the other hand, repetitive events are particular type of events that are repeated with a frequency specified by the User. For these type of events, the User will be able to provide informations about start and end time and location, but will not be able to select a global travel means. For each occurrence of the event, the User will be allowed to request travel options and choose a travel mean.

- **Transport booking**
  The System provides a way to book some of the travel means proposed during the schedule phase relying on third party services. The User will be automatically redirected to the app or to the site of the service supplier to complete the purchase. However, the System provides only an interface for these services: the User might be requested to sign up/log in and to provide payment information in order to buy a ticket or book a ride. All the operations which occur in this phase are totally performed out of the application.

2.3 User Characteristics

Travlendar+ is suitable for all kind of Users, with no age limit. We give for granted that both Users have access to Internet and are able to install and use the mobile application.
The characters are:

- *Guest:* a person who downloads and opens the app but still has to sign up. He/she cannot use any of the functions provided by *Travlendar+*.

- *User:* a registered User who has to log-in in order to access any feature of the application. He/she can retrieve his/her personal data from any device with the app installed just providing credentials.

- *Logged-in User:* a User who logged in to the application and can manage his personal schedule, edit personal information and preferences. In order to book travels, he/she might need to be registered on third-party services.

2.4 Constraints, Dependencies, Assumptions

Constraints

- Users are located in Milan.
- Users must have a smartphone equipped with an OS compatible with the application.
- Users must have Internet connection.
- The System must ask Users for the permission to acquire, store and process personal data. Therefore, the System must offer the possibility to the Users to delete their personal account and the associated data.

Dependencies

- The System will rely on external APIs to retrieve informations about travel means and associated ETA.
- Travel booking options are provided by external third party services.
- The System needs a DBMS in order to store and retrieve Users' data.

Assumptions

- [**D.1**] The given email is assumed to be correct.
- [**D.2**] The sent email is assumed to be correctly received.
- [**D.3**] The events informations provided by the user are correct.
- [**D.4**] Event time constraint are respected by the User.
- [**D.5**] The informations about the event are correct.
- [**D.6**] The informations about available mobility options and related travel time are correct.
- [**D.7**] If selected, the private means of transport are available without any delay.
- [**D.8**] The start location of the travel is known and correct.
- [**D.9**] All selected travel means specified in the preferences are potentially available to the User.

- [**D.10**] A minimum walking distance is assumed in order to reach every mean of transport. Such distance will not be taken into account unless significant with respect to the travel time, meaning unless it influences the ETA by more than a minute.
- [**D.11**] The User is registered to the service which offers the booking option.
- [**D.12**] The System internal clock time used to provide notifications is correct.

3 Specific Requirements

3.1 External Interface Requirements

The *Travlendar+* application is a mobile based application.
In the following section, a more detailed description of the application is given, in terms of hardware, software and communication interfaces.
Are also given some basic prototypes of the User interface through mockups.

3.1.1 User Interfaces

Aiming at being as simple and intuitive as possible, the application has a UI which follows the principles of flat design.

- **Logo** The logo of *Travlendar+*, which will also be the logo icon for the app on smartphones, is minimal but elaborate at the same time. Multiple rounded rectangle and squares are combined in a way that makes possible to distinguish both the letter 't' and the symbol '+'. This logo will also be shown during the application opening phase.

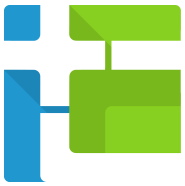


Figure 11: Icon

travlendar+

Figure 12: Logo

- **Sign In/Sign up**
  This form is the first thing the User sees after opening the app. It allows the User to log in, signup or recover the password if lost. The registration module requires three mandatory fields..

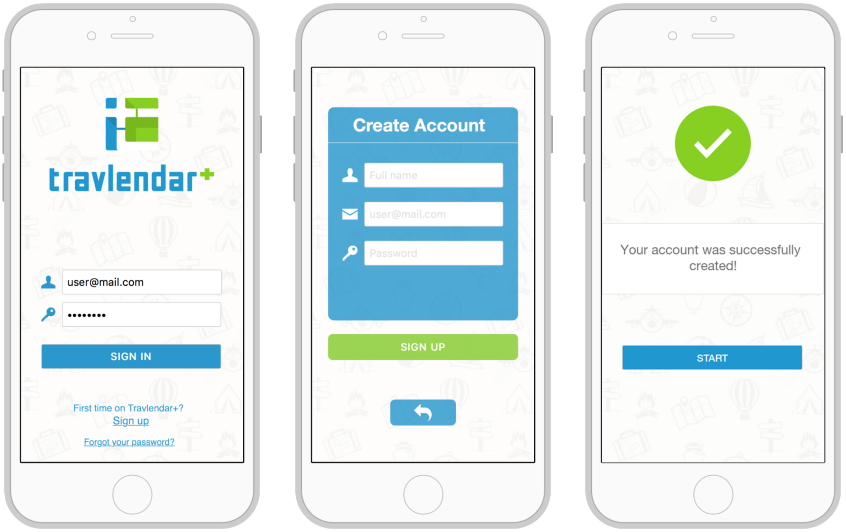


Figure 13: Sign in and sign up procedures

- **Home page**
  Contains all the events of the current day, which could be expanded and edited, and a button to add a new event for the current day.

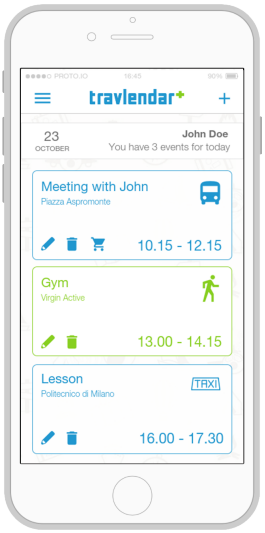


Figure 14: Daily schedule

- **User settings**
  Used to view and edit the profile picture, personal email address, username and password.

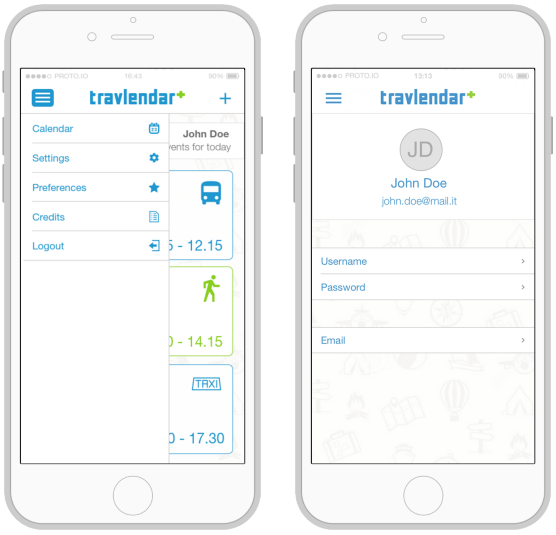


Figure 15: Menu and User settings

- **User preferences**
  In this page the User is able to activate or deactivate the available travel means and to set specific constraints for each of them.

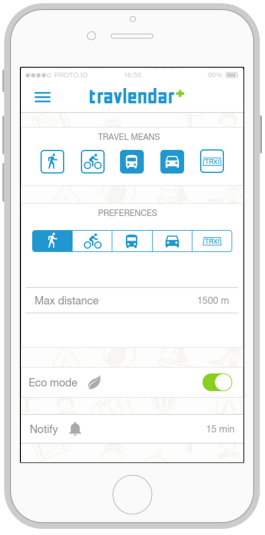


Figure 16: User preferences view

- **Calendar page**
  Displays the User calendar with a bullet for each day with at least one event scheduled.

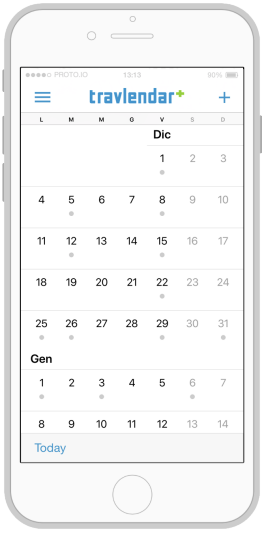


Figure 17: Calendar view

- **New/Edit event**
  Opened when an User wants to add a new event to the calendar or to edit an existent one. If fields are consistent and there is no conflict in the schedule a list of possible travel options is shown.

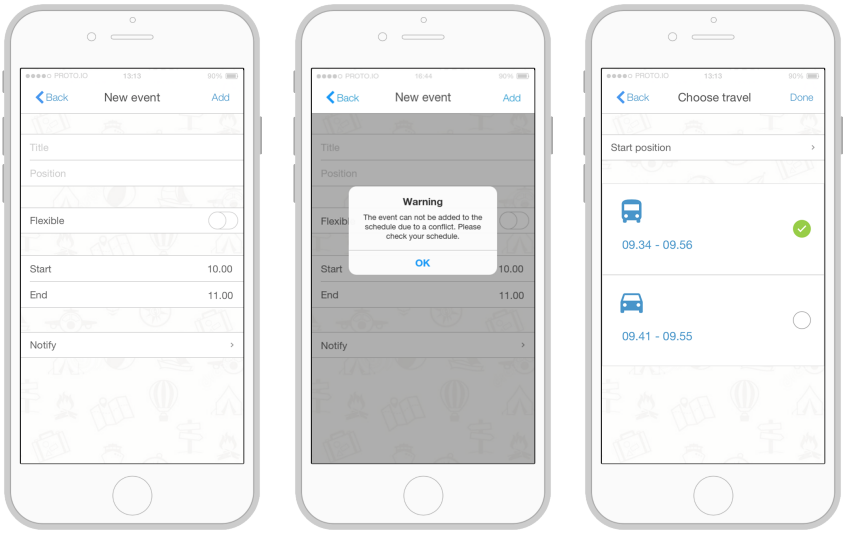


Figure 18: Add event procedure

3.1.2 Hardware Interfaces

The application is available for mobile devices that guarantee Internet access. The web application can be accessed by any device that provides a reasonably recent browser.

3.1.3 Software Interfaces

- Operating System: iOS, Android
- Web Browser
- Web Server application
- Development Frameworks
- DBMS
- Booking System: APIs for third party external services
- Mailing System: APIs to send emails to the User
- Mapping System: APIs for retrieve transport information

3.1.4 Communication Interfaces

The application will use HTTPS protocol for communication over the Internet and with the DBMS.

3.2 Functional Requirements

In the following section are explained the functional requirements of the application.

- [**R.1**] A visitor must be able to register. During the registration the System will ask to provide credentials.
- [**R.2**] The System must check if the Guest credentials are valid:
  - the username is not already taken by another registered User
  - the email is in the right format
  - the password has a minimum length.

  If credentials are correct, the System sends a confirmation email.

- [**R.3**] The System must store all User data such as personal information, credentials and schedule information.
- [**R.4**] The System must allow the User to log in using his/her personal credentials.
- [**R.5**] The System must allow the User to change username, only if the new username is not already in use by another User, email, only if the new email is in a correct format and password, only if the new password is different from the precedent and respects the minimum length.
- [**R.6**] The System must send a confirmation email if username, email or password is changed. The System must replace the old credentials with the new one.
- [**R.7**] The User must be allowed to create events, specifying:
  - The name of the event
  - The location of the event
  - The location from which the event will be reached
  - Start and end date of the event
  - Start and end time of the event
  - Notification option
- [**R.8**] The User must be allowed to edit or delete a specific event in his/her schedule.
- [**R.9**] The System allows the User to provide optional event information. This informations are:
  - Category type of the event
  - Brief description of the event
- [**R.10**] The System must allow the User to view all the events for a window of time.
- [**R.11**] The System must check if the event created or edited by the User is feasible.
- [**R.12**] The System must compute travel time between appointments.

- [**R.13**] The System must guarantee a feasible schedule, that is, the User is able to move from an appointment to another in time. The System must warn the User in case of unfeasible event preventing the creation.
- [**R.14**] The System allows the User to add or edit an event only if it's not in conflict with other events already existent.
- [**R.15**] The System must allow the User to select a specific event.
- [**R.16**] The System must compute the best mobility options taking into account User preferences. If specified, the System must calculate the feasible combination of transportation that minimize carbon footprint.
- [**R.17**] The System must allow the User to choose one of the mobility option during the event creation and the editing phase of the event.
- [**R.18**] The System allows the User to define specific constraint for each travel means, that are:
  - The maximum distance reachable for each mobility option.
  - The minimum distance of travel necessary for a specific mean of transport to be take into account.
- [**R.19**] The System must allow the User to be notified a specific time before any event.
- [**R.20**] The System must allow the User to select the *Eco Mode* in order to minimize carbon footprint.
- [**R.21**] The System must allow the User to create flexible events and add additional informations that are:
  - Time window in which the event could be created
  - Duration of the event
- [**R.22**] The System must allow the User to create repetitive events, adding information concerning:
  - start/end time
  - location of the event
  - frequency of repetition.

- [**R.23**] The System must check the feasibility of flexible events. The System must adapt these events in the schedule only if they are not in conflict with other events already in the schedule.
- [**R.24**] The System must check the feasibility of repetitive events, taking into account an estimation of the travel time computed at the moment of the creation of the event. The System must warn the User in case of conflicts for some of the day specified: the User will be allowed to add the event only in the day with no conflicts or to discard the event creation.
- [**R.25**] For each occurrence of a repetitive event, the User will be allowed to request travel options and choose a travel mean.
- [**R.26**] The System must allow the User to select a specific travel means.
- [**R.27**] The System must provide an interface for third party services allowing the User to authenticate with the service.
- [**R.28**] The System must allow to activate notification and setting its time.
- [**R.29**] The System must activate a ring at the time selected by the User.

3.3 Goals, Requirements and Domain Assumptions

In the following section functional requirements and domain assumptions are grouped under each goal.

- [**G.1**] **The System allows the User to access the functionalities of the application from different locations and devices. The data has to be coherent across different devices.**

  Requirements

  – [**R.1**] A visitor must be able to register. During the registration the System will ask to provide credentials.

  – [**R.2**] The System must check if the Guest credentials are valid:

    * the username is not already taken by another registered User
    * the email is in the right format
    * the password has a minimum length.

    If credentials are correct, the System sends a confirmation email.

- [**R.3**] The System must store all User data such as personal information, credentials and schedule information.
- [**R.4**] The System must allow the User to log in using his/her personal credentials.
- [**R.5**] The System must allow the User to change username, only if the new username is not already in use by another User, email, only if the new email is in a correct format and password, only if the new password is different from the precedent and respects the minimum length.
- [**R.6**] The System must send a confirmation email if username, email or password is changed. The System must replace the old credentials with the new one.

Domain assumptions

- [**D.1**] The given email is assumed to be correct.
- [**D.2**] The sent mail is assumed to be correctly received.
- [**D.3**] The Storage System is reliable.

- [**G.2**] **The System allows the User to manage meetings in his/her schedule.**

Requirements

- [**R.4**]The System must allow the User to log in using his/her personal credentials.
- [**R.7**] The User must be allowed to create events, specifying:
  - The name of the event
  - The location of the event
  - The location from which the event will be reached
  - Start and end date of the event
  - Start and end time of the event
  - Notification option
- [**R.8**] The User must be allowed to edit or delete a specific event in his/her schedule.
- [**R.9**] The System allows the User to provide optional event information. This informations are:
  - Category type of the event
  - Brief description of the event

- [**R.10**] The System must allow the User to view all the events for a window of time.
- [**R.11**] The System must check if the event created or edited by the User is feasible.

  Domain assumptions

- [**D.3**] The events informations provided by the user are correct.

- **[G.3] The System allows the User to reach every meeting on time.**

  Requirements

  - [**R.12**] The System must compute travel time between appointments.
  - [**R.13**] The System must guarantee a feasible schedule, that is, the User is able to move from an appointment to another in time. The System must warn the User in case of unfeasible event preventing the creation.
  - [**R.14**] The System allows the User to add or edit an event only if it's not in conflict with other events already existent.

  Domain assumptions

  - [**D.4**] Event time constraint are respected by the User.
  - [**D.5**] The informations about the event are correct.

- **[G.4] The System allows the User to select or edit a travel means to reach an event.**

  Requirements

  - [**R.15**] The System must allow the User to select a specific event.
  - [**R.16**] The System must compute the best mobility options taking into account User preferences. If specified, the System must calculate the feasible combination of transportation that minimize carbon footprint.
  - [**R.17**] The System must allow the User to choose one of the mobility option during the event creation and the editing phase of the event.

  Domain assumptions

  - [**D.6**] The informations about available mobility options and related travel time are correct.

- [**D.7**] If selected, the private means of transport are available without any delay.
- [**D.8**] The start location of the travel is known and correct.

- **[G.5] The System allows the User to set preferences.**

  Requirements

  - [**R.4**] The System must allow the User to login using his/her personal credentials.
  - [**R.18**] The System allows the User to define specific constraint for each travel means, that are:
    * The maximum distance reachable for each mobility option.
    * The minimum distance of travel necessary for a specific mean of transport to be take into account.
  - [**R.19**] The System must allow the User to be notified a specific time before any event.
  - [**R.20**] The System must allow the User to select the *Eco Mode* in order to minimize carbon footprint.

  Domain assumption

  - [**D.9**] All selected travel means specified in the preferences are potentially available to the User.
  - [**D.10**] A minimum walking distance is assumed in order to reach every mean of transport. Such distance will not be taken into account unless significant with respect to the travel time, meaning unless it influences the ETA by more than a minute.

- [**G.6**] The System allows the User to create flexible events and repetitive events.

  Requirements

  - [**R.4**] The System must allow the User to login using his/her personal credentials.
  - [**R.21**] The System must allow the User to create flexible events and add additional informations that are:
    * Time window in which the event could be created
    * Duration of the event
  - [**R.22**] The System must allow the User to create repetitive events, adding information concerning:

    - start/end time
    - location of the event
    - frequency of repetition.
  - [**R.23**] The System must check the feasibility of flexible events. The System must adapt these events in the schedule only if they are not in conflict with other events already in the schedule.
  - [**R.24**] The System must check the feasibility of repetitive events, taking into account an estimation of the travel time computed at the moment of the creation of the event. The System must warn the User in case of conflicts for some of the day specified: the User will be allowed to add the event only in the day with no conflicts or to discard the event creation.
  - [**R.25**] For each occurrence of a repetitive event, the User will be allowed to request travel options and choose a travel mean.

  Domain assumption

  - [**D.4**] Event time constraint are respected by the User.

- [**G.7**] **The System allows the User to book transportation for a travel.**

  Requirements

  - [**R.4**] The System must allow the User to login using his/her personal credentials.
  - [**R.26**] The System must allow the User to select a specific travel means.
  - [**R.27**] The System must provide an interface for third party services allowing the User to authenticate with the service.

  Domain assumptions

  - [**D.11**] The User is registered to the service which offers the booking option.

- [**G.8**] **The System allows the User to be notified before the occurrence of an event.**

  Requirements

  - [**R.4**] The System must allow the User to login using his/her credentials.

– [**R.28**] The System must allow to activate notification and setting its time.

– [**R.29**] The System must activate a ring at the time selected by the User.

Domain assumptions

– [**D.12**] The System internal clock time used to provide notifications is correct.

3.4 Performance Requirements

Travlendar+ is structured in a three-tier architecture. All the application logic is built in the application servers: the backend performances should guarantee the proper operation in case of traffic peaks and manage multiple Users.

3.5 Design Constraints

3.5.1 Standards Compliance

All Informations concerning locations are given in form of Latitude and Longitude degrees.

3.5.2 Hardware Limitations

The application should be able to run, at least, under the following conditions:

- 3G connections at 2Mb/s
- 50 MB of space
- 2 GB of RAM

3.6 Software Attributes

3.6.1 Reliability

The System should guarantees a 99% of reliability: its components must have a failure rate that guarantees this goal.

3.6.2 Availability

The System guarantees an high availability, due to the expected high reliability, in order to offer an ideal 24/7 service.

3.6.3 Security

User credentials and data will be stored in a DBMS that should guarantee an high security. For what concerns the User credential, in order to provide this result, the passwords stored in the DBMS are salted and hashed.
As already mentioned, the System uses HTTPS protocol to communicate with all the services, in order to guarantee protection of the privacy and integrity of the exchanged data.
The payment security for the booking functionality is guaranteed by third-party services and the System is not responsible in case of damage.

3.6.4 Maintainability

The System will have a modular architecture: each component is organized in a hierarchical way in order to speed up the maintenance in case of failure. This structure is also convenient for further refinements of single components.

3.6.5 Portability

The application is developed in terms of Web Application and native application for iOS and Android. Furthermore, the three-tier architecture of the System allows to define a backend in which all the business logic is built, ensuring an high level of portability.

4 Formal Analysis Using Alloy

In this section a model of the World is provided by the Alloy tool.

4.1 Model

```
open util/boolean

---------------- AVAILABLE PREFERENCES ----------------

sig publicPreference extends Preference {
maxCost : Int,
maxChanges : Int
} { maxCost > 0 and maxChanges ≥ 0}

sig footPreference extends Preference {
maxDistance: Int
} {maxDistance > 0}

sig bikePreference extends Preference {
maxDistance: Int
} {maxDistance > 0}

sig taxiPreference extends Preference {
maxCost: Int
} {maxCost > 0}

sig carPreference extends Preference {
minDistance: Int
} {minDistance > 0}

sig bikeSharingPreference extends Preference {
maxDistance: Int,
maxCost: Int
} {maxCost > 0 and maxDistance > 0}

sig carSharingPreference extends Preference {
minDistance: Int,
maxCost: Int
} {maxCost > 0 and minDistance > 0}

---------------- AVAILABLE TRANSPORTS ----------------

-- an instance of transport is intended as additional information about a
    ↪ specific travel once the type of mean is defined

sig Foot extends Transport {
distance : Int
} { distance > 0 }

sig Taxi extends Transport {
cost : Int
} { cost > 0 }

sig Car extends Transport {
distance : Int
} { distance > 0 }

sig Public extends Transport {
```

```
cost : Int,
nChanges: Int
} { cost > 0 and nChanges ≥ 0}

sig Bike extends Transport {
distance : Int
} { distance > 0 }

sig carSharing extends Transport {
distance: Int,
cost: Int
} { cost > 0 and distance > 0}

sig bikeSharing extends Transport  {
distance,
cost: Int,
} { cost > 0 and distance > 0}

---------------- MODEL SIGNATURES ----------------

abstract sig Preference {}
abstract sig Transport {}

sig Booking {}

sig User {
calendar: set Schedule,
constraints: set Preference,
id: one Int
} { id > 0 }

sig Schedule {
scheduleEvents: set Event,
scheduleTravels: set Travel
} {
-- each schedule has 1 starting event i.e. no travel has this event as a
    ↪ destination
-- and 1 final event i.e. no travel has this event as a source, these events
    ↪  can coincide.
-- all other events are connected by travels
(one e: scheduleEvents | no t: scheduleTravels | t.to = e)      and (one e:
    ↪ scheduleEvents | no t: scheduleTravels | t.from = e)
}

sig Event {
start: one Int,
end: one Int,
active: one Bool
} { start > 0 and end > 0 and start < end }

sig Travel {
from: one Event,
to: one Event,
start: one Int,
end: one Int,
mean: one Transport,
travelBooking: lone Booking
} { start > 0 and end > 0 and start < end}

-- each preference is in the constraints of 1 user
fact eachPreferenceBelongsToAUser {
all p: Preference | one u: User  | p in u.constraints
```

```
}

-- the id associated with each user is unique
fact userIdsAreUnique {
no disjoint u1, u2 : User | u1.id = u2.id
}

-- each users can specify one constraint/preference of each type
fact userHasOnePreferencePerType {
all u: User {
lone p: Preference | p in bikePreference and p in u.constraints
lone p: Preference | p in footPreference and p in u.constraints
lone p: Preference | p in publicPreference and p in u.constraints
lone p: Preference | p in bikeSharingPreference and p in u.constraints
lone p: Preference | p in carSharingPreference and p in u.constraints
lone p: Preference | p in carPreference and p in u.constraints
lone p: Preference | p in taxiPreference and p in u.constraints
}
}

-- the preferences of each user must be respected in the model
fact preferencesMustBeRespected {
no u: User {
all s: Schedule {
all t: Travel {
some p: publicPreference, pt: Public | pt.nChanges > p.maxChanges and p in u
    ↪ .constraints and t in s.scheduleTravels and t.mean = pt
some p: publicPreference, pt: Public | pt.cost > p.maxCost and p in u.
    ↪ constraints and t in s.scheduleTravels and t.mean = pt
some p: footPreference, ft: Foot | ft.distance > p.maxDistance and p in u.
    ↪ constraints and t in s.scheduleTravels and t.mean = ft
some p: bikePreference, bt : Bike | bt.distance > p.maxDistance and p in u.
    ↪ constraints and t in s.scheduleTravels and t.mean = bt
some p: taxiPreference, tt: Taxi | tt.cost > p.maxCost and p in u.
    ↪ constraints and t in s.scheduleTravels and t.mean = tt
some p: carPreference, ct : Car | ct.distance < p.minDistance and p in u.
    ↪ constraints and t in s.scheduleTravels and t.mean = ct
some p: carSharingPreference, st : carSharing | st.cost > p.maxCost and p in
    ↪  u.constraints and t in s.scheduleTravels and t.mean = st
some p: carSharingPreference, st : carSharing | st.distance < p.minDistance
    ↪ and p in u.constraints and t in s.scheduleTravels and t.mean = st
some p: bikeSharingPreference, st : bikeSharing | st.distance > p.
    ↪ maxDistance and p in u.constraints and t in s.scheduleTravels and t.
    ↪ mean = st
some p: bikeSharingPreference, st : bikeSharing | st.cost > p.maxCost and p
    ↪ in u.constraints and t in s.scheduleTravels and t.mean = st
}
}
}
}

-- each schedule is in the calendar of 1 user
fact eachScheduleBelongsToAUser {
all s:Schedule | one u: User | s in u.calendar
}

-- each active event is in 1 schedule
fact eachEventBelongsToASchedule {
all e: Event | one s: Schedule | e in s.scheduleEvents or (e.active = False
    ↪ and not e in s.scheduleEvents)
}
```

```
-- booking is available for public transportation, taxi and sharing services
fact bookingServices {
all t: Travel | all b : Booking | t.travelBooking = b implies t.mean in Taxi
    ↪  + Public + carSharing + bikeSharing
}

-- each travel is in 1 schedule
fact eachTravelBelongsToASchedule {
all t: Travel | one s: Schedule | t in s.scheduleTravels
}

-- each booking referes to 1 travel
fact eachBookingBelongsToATravel {
all b: Booking | one t: Travel | t.travelBooking = b
}

-- each instance of transport is associated with a travel
fact eachTransportBelongsToATravel {
all tt: Transport | one t: Travel | t.mean = tt
}

-- inactive events do not belong in any schedule
fact inactiveEventsdoNotBelongInAnySchedule {
all s: Schedule | all e: Event | e in s.scheduleEvents implies e.active =
    ↪ True
}

-- for any given travel there are 2 events, source and destination
-- that respectively end before the start of the travel and start after the
    ↪ start of the travel
fact travelIsBetweenEventsTimes {
no t: Travel | t.start < t.from.end or t.end > t.to.start
}

-- each travel must have a source and a destination
fact travelHasToAndFrom {
all t: Travel | some disjoint e1, e2: Event | t.to = e1 and t.from = e2
}

-- no travel can connect events from 2 different schedule
fact noTravelAcrossSchedules {
all t: Travel | all s: Schedule | t.to in s.scheduleEvents implies t.from in
    ↪  s.scheduleEvents and t in s.scheduleTravels
}

-- no travel can connect to or from inactive events
fact noTravelToOrFromInactive {
no t: Travel | t.from.active = False or t.to.active = False
}

-- this means that the instances of the transport are different not the type
    ↪  of transport
fact differentTravelHaveDifferentMean {
no disjoint t1, t2 : Travel | t1.mean = t2.mean
}

-- for each 2 events the start or end time can not coincide
fact scheduleEventsDoNotEndOrStartAtTheSameTime {
all s: Schedule | all disjoint e1, e2: Event |
(e1 in s.scheduleEvents and e2 in s.scheduleEvents) implies (e1.start ≠ e2.
    ↪ start and e1.end ≠ e2.end)
}
```

```
-- events do not intersect
fact noIntersectingEvents {
all s: Schedule | all disjoint e1, e2: Event |
(e1 in s.scheduleEvents and e2 in s.scheduleEvents and e1.start < e2.start)
    ↪ implies e1.end < e2.start
}

-- there is only one travel that has a certain event as source or as a
    ↪ destination
fact noDoubleTravelPerEvent {
no disjoint t1, t2 : Travel | t1.from = t2.from or  t1.to = t2.to
}
---------------- ASSERTIONS ----------------

-- the event that is not the destination of any travel (starting
assert noEventBeforeFirstAndAfterLastEvent {
all s: Schedule | no disjoint e1, e2 : Event |
isFirstOfSchedule[e1,s] and e2.start < e1.end and e2 + e1 in s.
    ↪ scheduleEvents or
isLastOfSchedule[e1,s] and e2.end > e1.start and e2 + e1 in s.scheduleEvents
}

-- there are no intersection between schedules in terms of travels and
    ↪ events
assert allSchedulesAreDisjoint {
all disjoint s1, s2: Schedule | no e: Event, t:Travel | (e in s1.
    ↪ scheduleEvents and e in s2.scheduleEvents)
or (t in s1.scheduleTravels and t in s2.scheduleTravels)
}

-- a travel only connects events in chronological order
assert cannotTravelInThePast{
all s: Schedule | all t: Travel | no e1, e2: Event | (e1+e2 in s.
    ↪ scheduleEvents and t.to = e1 and t.from = e2)
and (e2.end > e1.start)
}

---------------- PREDICATES ----------------

pred show {}

pred isLastOfSchedule[e: Event, s: Schedule] {
no t : Travel | t.from = e and e in s.scheduleEvents and t in s.
    ↪ scheduleTravels
}

pred isFirstOfSchedule[e: Event, s: Schedule] {
no t : Travel | t.to = e and e in s.scheduleEvents and t in s.
    ↪ scheduleTravels
}

pred addEvent[e: Event, s: Schedule] {
s.scheduleEvents = s.scheduleEvents + e
}

pred addBooking[t, t': Travel, new_b: Booking, s: Schedule] {
t'.travelBooking = new_b
t'.from = t.from
t'.to = t.to
t'.start = t.start
t'.end = t.end
```

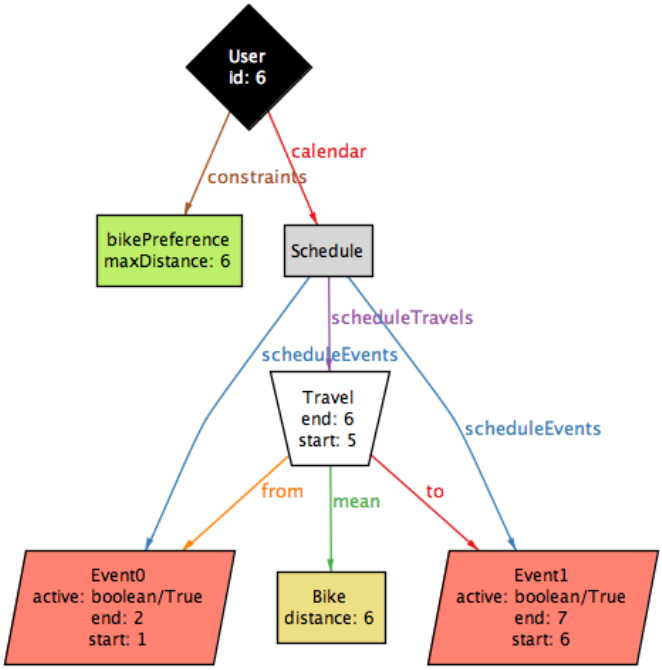
```
t'.mean = t.mean
s.scheduleTravels = s.scheduleTravels - t + t'
t + t' in s.scheduleTravels
}

pred changeEventTime[e, e' : Event, s: Schedule, new_end, new_start : Int] {
e'.start = new_start
e'.end = new_end
e'.active = e.active
s.scheduleEvents = s.scheduleEvents - e + e'
}

pred changeTravelTransport[t, t': Travel, s: Schedule, new_mean: Transport]
    ↪ {
t'.travelBooking = t.travelBooking
t'.from = t.from
t'.to = t.to
t'.start = t.start
t'.end = t.end
t'.mean = new_mean
s.scheduleTravels = s.scheduleTravels - t + t'
t + t' in s.scheduleTravels
}

---------------- CHECK ASSERTIONS ----------------

check noEventBeforeFirstAndAfterLastEvent for  10 Event, 10 Travel, 10 User,
    ↪  10 Schedule, 10 Transport, 10 Booking, 6 Preference, 4 Int
check allSchedulesAreDisjoint for  10 Event, 10 Travel, 10 User, 10 Schedule
    ↪ , 10 Transport, 10 Booking, 6 Preference, 4 Int
check cannotTravelInThePast for  10 Event, 10 Travel, 10 User, 10 Schedule,
    ↪ 10 Transport, 10 Booking, 6 Preference, 4 Int

---------------- RUN PREDICATES ----------------

run show for exactly 4 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪  Booking,  5 Preference, 4 Int
run addBooking for 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪ Booking,  5 Preference, 4 Int
run addEvent for 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪ Booking,  5 Preference, 4 Int
run changeEventTime for 10 Event, 10 Travel, 2 User, 2 Schedule, 10
    ↪ Transport, 2 Booking,  5 Preference, 4 Int
run changeTravelTransport for 10 Event, 10 Travel, 2 User, 2 Schedule, 10
    ↪ Transport, 2 Booking,  5 Preference, 4 Int
```

4.2 Model Checks

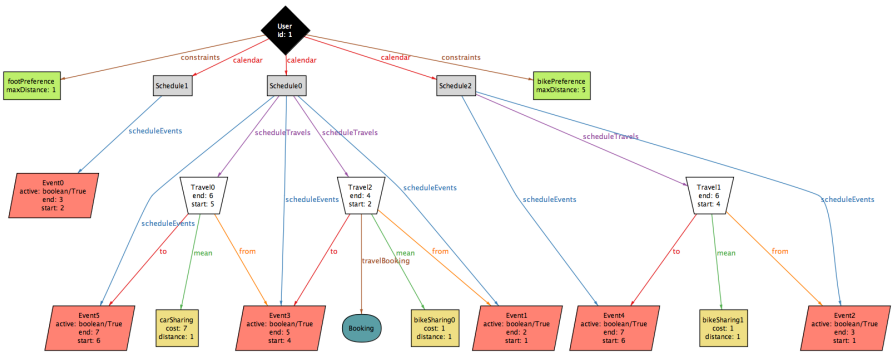
```
Executing "Check noEventBeforeFirstAndAfterLastEvent for 4 int, 10 Event, 10 Travel, 10 User, 10 Schedule, 10 Transport, 10 Booking, 6 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   120122 vars. 4732 primary vars. 273595 clauses. 1320ms.
   No counterexample found. Assertion may be valid. 7358ms.

Executing "Check allSchedulesAreDisjoint for 4 int, 10 Event, 10 Travel, 10 User, 10 Schedule, 10 Transport, 10 Booking, 6 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   118632 vars. 4742 primary vars. 268755 clauses. 943ms.
   No counterexample found. Assertion may be valid. 46ms.

Executing "Check cannotTravelInThePast for 4 int, 10 Event, 10 Travel, 10 User, 10 Schedule, 10 Transport, 10 Booking, 6 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   119172 vars. 4742 primary vars. 271107 clauses. 888ms.
   No counterexample found. Assertion may be valid. 135ms.

Executing "Run show for 4 int, exactly 4 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2 Booking, 5 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   63215 vars. 3355 primary vars. 158276 clauses. 224ms.
   Instance found. Predicate is consistent. 177ms.

Executing "Run addBooking for 4 int, 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2 Booking, 5 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   80411 vars. 3845 primary vars. 201043 clauses. 254ms.
   Instance found. Predicate is consistent. 755ms.

Executing "Run addEvent for 4 int, 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2 Booking, 5 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   78741 vars. 3833 primary vars. 195347 clauses. 275ms.
   Instance found. Predicate is consistent. 1421ms.

Executing "Run changeEventTime for 4 int, 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2 Booking, 5 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   79409 vars. 3875 primary vars. 197312 clauses. 277ms.
   Instance found. Predicate is consistent. 141ms.

Executing "Run changeTravelTransport for 4 int, 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2 Booking, 5 Preference"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   80387 vars. 3853 primary vars. 200827 clauses. 271ms.
   Instance found. Predicate is consistent. 2374ms.

8 commands were executed. The results are:
   #1: No counterexample found. noEventBeforeFirstAndAfterLastEvent may be valid.
   #2: No counterexample found. allSchedulesAreDisjoint may be valid.
   #3: No counterexample found. cannotTravelInThePast may be valid.
   #4: Instance found. show is consistent.
   #5: Instance found. addBooking is consistent.
   #6: Instance found. addEvent is consistent.
   #7: Instance found. changeEventTime is consistent.
   #8: Instance found. changeTravelTransport is consistent.
```

4.3 Generated Worlds

```
pred show {
#Schedule = 1
#User = 1
#Preference = 1
#Booking = 0
}

run show for exactly 2 Event, 10 Travel, 2 User, 3 Schedule, 10 Transport, 2
    ↪ Booking, 5 Preference, 4 Int
```

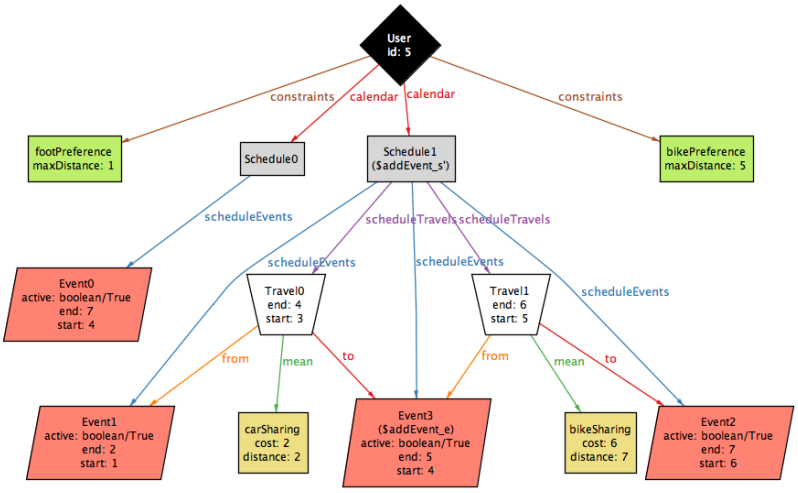


```
pred show {
#Schedule = 3
#User = 1
#Preference = 2
#Booking = 1
}

run show for exactly 6 Event, 10 Travel, 2 User, 3 Schedule, 10 Transport, 2
    ↪ Booking, 5 Preference, 4 Int
```

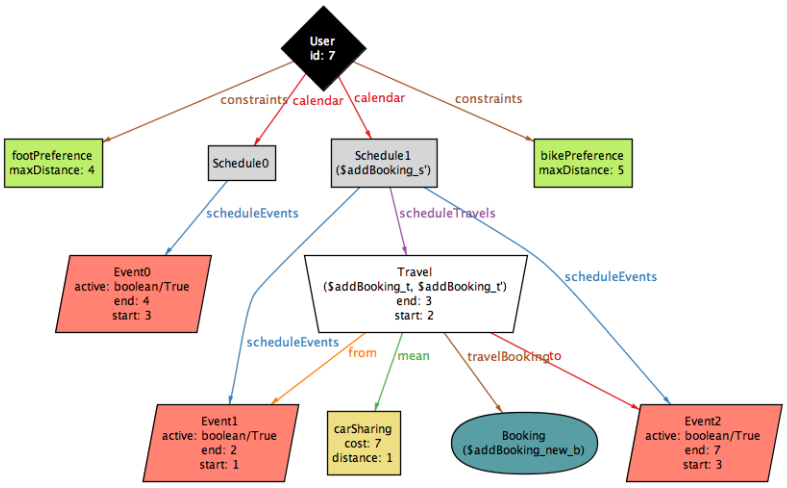



```
pred show {
#Schedule = 2
#User = 2
#Preference = 4
#Booking = 1
}

run show for exactly 6 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪ Booking, 5 Preference, 4 Int
```


```
pred addEvent[e: Event, s: Schedule] {
s.scheduleEvents = s.scheduleEvents + e
}

run addEvent for 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪ Booking, 5 Preference, 4 Int
```

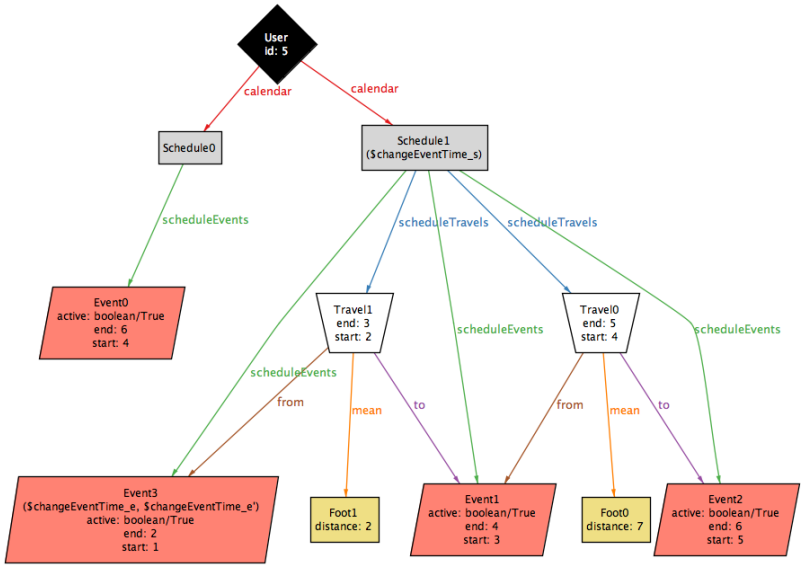



```
pred addBooking[t, t': Travel, new_b: Booking, s: Schedule] {
t'.travelBooking = new_b
t'.from = t.from
t'.to = t.to
t'.start = t.start
t'.end = t.end
t'.mean = t.mean
s.scheduleTravels = s.scheduleTravels - t + t'
t + t' in s.scheduleTravels
}

run addBooking for 10 Event, 10 Travel, 2 User, 2 Schedule, 10 Transport, 2
    ↪ Booking,  5 Preference, 4 Int
```

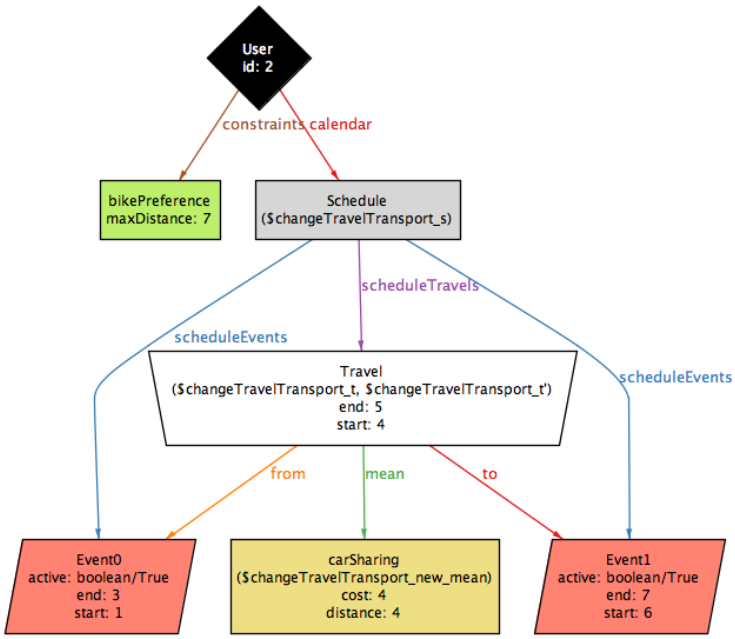



```
pred changeEventTime[e, e' : Event, s: Schedule, new_end, new_start : Int] {
e'.start = new_start
e'.end = new_end
e'.active = e.active
```

```
s.scheduleEvents = s.scheduleEvents - e + e'
e + e' in s.scheduleEvents
}

run changeEventTime for 10 Event, 10 Travel, 2 User, 2 Schedule, 10
    ↪ Transport, 2 Booking,  5 Preference, 4 Int
```




```
pred changeTravelTransport[t, t': Travel, s: Schedule, new_mean: Transport]
    ↪ {
t'.travelBooking = t.travelBooking
t'.from = t.from
t'.to = t.to
t'.start = t.start
t'.end = t.end
t'.mean = new_mean
s.scheduleTravels = s.scheduleTravels - t + t'
t + t' in s.scheduleTravels
}

run changeTravelTransport for 10 Event, 10 Travel, 2 User, 2 Schedule, 10
    ↪ Transport, 2 Booking,  5 Preference, 4 Int
```


User
id: 2
constraints calendar
bikePreference
maxDistance: 7
Schedule
($changeTravelTransport_s)
scheduleTravels
scheduleEvents
scheduleEvents
Travel
($changeTravelTransport_t, $changeTravelTransport_t')
end: 5
start: 4
from
mean
to
Event0
active: boolean/True
end: 3
start: 1
carSharing
($changeTravelTransport_new_mean)
cost: 4
distance: 4
Event1
active: boolean/True
end: 7
start: 6

5 Effort Spent

The effort spent from each member of the team to build the RASD can be summarized as follow:

Guglielmo Menchetti

Date	**Task**	**Hours**
02/10/17	First meeting	4
03/10/17	Github setup	1,5
04/10/17	Purpose and Scope	3,5
06/10/17	Product Functions	2,5
09/10/17	Design Constraints	2
10/10/17	Mapping Goals and Functional Requirements	4,5
12/10/17	Goals review	2,5
14/10/17	External Interface Requirements	2
15/10/17	Mid-phase meeting	3,5
19/10/17	Software Attributes	1,5
20/10/17	Requirements review	2,5
22/10/17	Domain Assumptions list	3
23/10/17	Introduction review	3
24/10/17	Alloy and Requirements	4
26/10/17	Global review	4
27/10/17	Alloy refinements	3
		Total
		47

Lorenzo Norcini

Date	Task	Hours
02/10/17	First meeting	4
03/10/17	Github setup	1,5
04/10/17	Purpose and Scope	3,5
07/10/17	Class Diagram	2,5
09/10/17	Performance Requirements	1,5
10/10/17	Mapping Goals and Functional Requirements	4,5
11/10/17	State Chart Diagrams	3
13/10/17	Sequence and Use Case Diagrams	4
15/10/17	Mid-phase meeting	3,5
17/10/17	Alloy first modeling	3
18/10/17	Requirements review	2,5
19/10/17	Software Attributes	1,5
23/10/17	Introduction review	3
24/10/17	Alloy modeling	4
26/10/17	Global review	4
27/10/17	Alloy modeling and refinements	3,5
		Total
		49,5

Tommaso Scarlatti

Date	Task	Hours
02/10/17	First meeting	4
03/10/17	Github setup	1,5
04/10/17	Purpose and Scope	3,5
06/10/17	Overview, Definitions and Acronyms	2
09/10/17	User Characteristics	1,5
10/10/17	Mapping Goals and Functional Requirements	4,5
12/10/17	Logo and Reference Documents	1,5
15/10/17	Mid-phase meeting	3,5
16/10/17	User Interface	2
18/10/17	User Interface	3,5
19/10/17	Software Attributes	1,5
20/10/17	Goals review and list	3,5
23/10/17	Introduction review	3
24/10/17	Use Case Templates	3,5
25/10/17	User Interface refinements	3
26/10/17	Global review	3
27/10/17	Further refinements	1,5
		Total
		46,5