Politecnico di Milano

A.Y. 2017/2018

Software Engineering 2: ***TravIendar+***

Requirements Analysis and Specification Document

Matteo Biasielli - Emilio Capo - Mattia Di Fatta

v. 1.1

Table of Contents

1. Introduction

1.1 Document purpose

This document focuses on the requirements analysis for the project Travlendar+. The application's purpose is to support users in handling out one of the most difficult nowadays' challenges: organization. No previous versions of this application were developed.
This document is meant to be a reference for any person who has an interest in the project. This includes, but is not limited to, development team members, stakeholders and end users.

1.2 Description of the problem

The aim of the project is to create an all-in-one system that unites services that are nowadays offered by various different applications (e.g. Calendar, Travel Scheduler). In order to use Travlendar+, final users must be registered and logged in.
Users should be able to schedule their activities directly through the application and, by taking into account travelling times, constraints and preferences expressed by the user, Travlendar+ should:

- Identify the best mobility option;
- Support the user in buying public transport tickets, if necessary;
- Locate the nearest car or bike sharing, if they represent the best solution;
- Warn the user when a place can't be reached in the available time.

In general, Travlendar+ should make it easier to organize complex schedules, by finding the best compromises between time optimization and the users' needs and preferences.

1.3 Actors

- **Guest**: a person that is not yet registered or logged in. This actor can only see the main page of Travlendar+, where he's asked to log in or start a registration procedure.
- **User**: a person that is registered and logged in. This type of actor can access to all the features offered by Travlendar+.
- **Third part systems**: external systems that, through the use of APIs, Travlendar+ can use to provide the services it intends to offer.
  (e.g. Google Maps' APIs can be used to calculate the time needed to move from a place to another by car, bike, public transport system and on foot).
  (e.g. a car sharing system's APIs can be used to locate the available cars and propose to the user to use a car sharing service)

1.4 Goals

Travlendar+ features should match with the following goals:

- [G1] Users should be able to use the system properly.
- [G2] Allow users to schedule an activity.
- [G3] Allow users to set constraints and preferences.
- [G4] Simplify procedures and reduce the number of operations that a user has to do to schedule a new activity.
- [G5] Make sure users are on time at their scheduled appointments or that they're warned when they can't be on time.

1.5 Definitions, Acronyms, Abbreviations

1.5.1 Definitions

- **User**: actor that is using the application and may want to access all functionalities.
- **Application**: with the term application we are talking about the desktop version, the website and mobile version of the Travlendar+ system.
- **Scheduling**: action performed by a user that is adding a new activity to his personal calendar.
- **Flexible Activity**: An activity with starting and ending time larger than the duration.
- **Fixed Activity**: An activity with fixed starting and ending time.

1.5.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **UI:** User Interface
- **API:** Application programming interface
- **UXD**: User Experience Diagram
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System

1.5.3 Abbreviations

- **[Gn]:** the n-th goal
- **[Rn]:** the n-th requirement
- **[NFRn]:** the n-th non-functional requirement
- **[An]:** the n-th assumption
- **[Cn]:** the n-th constraint
- **[BPMNn]:** the n-th BPMN diagram
- **[UCn]:** the n-th use case table
- **[UIn**]: the n-th user interface example
- **[PWn]:** the n-th possible world

1.6 Reference Documents

- Mandatory project assignments for the A.Y. 2017/2018 available on the beep's page of the Software Engineering 2 course.
- Projects examples and other documents available on the beep's page of the Software Engineering 2 course.

1.7 Document Structure

- **Introduction:** This is the very first part of the document.
  In this sectionit's possible to retrieve general information about the project and its goals and about the system that is going to be described with more details in the next sections. For the sake of simplicity and to avoid any possible ambiguity, Acronyms, Definitions and Abbreviations that will be used in the whole document have been specified here.
- **Overall Description:** This section will contain a more detailed explanation of the product perspective, that can be useful to visualize and contextualize the project we're working on. It also highlights the several assumptions and constraints related to the project. Further information about the application's functions, the user and the requirements will be clarified.
- **Specific Requirements:** This section contains more details and examples about the Graphic User Interface and specifies which are the Communication Interfaces that must be supported by the end-user devices in order to make the application work properly.
  In addition, functional requirements will be defined with more details and they will be mapped into goals.
  Several UML diagrams will be displayed in this section.
- **Formal Analysis using Alloy:** The source code of an Alloy 4.2 specification of the system and analysis of some possible worlds.
- **Effort Spent:** Information about the number of hours each group member has spent working on the RASD document.

1.8 Revision History

- **v. 0.1 [05 Oct 2017]**: added the whole "Introduction" section.
- **v. 0.2 [07 Oct 2017]**: added part of the "Overall Description" section.
- **v. 0.3 [08 Oct 2017]**: completed the "Overall Description" section.
- **v. 0.4 [09 Oct 2017]:** general group revision of the first two sections.
- **v. 0.5 [14 Oct 2017]:** Added specific requirements.
- **v. 0.6 [16 Oct 2017]:** Added UI, BPMN, Various UML charts
- **v. 0.7 [17 Oct 2017]:** Finished Specific Requirements
- **v. 0.8 [23 Oct 2017]:** Added Alloy
- **v. 1.0 [28 Oct 2017]:** Finalized RASD
- **v. 1.1 [06 Nov 2017]:** Added assumption [A13]

2.0 Overall Description

This section includes a summary of the major functions provided by the system, the user characteristics, the constraints and the assumptions over the domain.

2.1 Product Perspective

Since the application can be used both on desktop computers and mobile devices and since a user may want to access his calendar from both kind of devices at the same time, data cannot be stored locally. The system we're going to develop will then consist of an application for the end user and an application for the central server. On the server, users' data, preferences and schedules will be stored in a Database.
Furthermore, our product needs to be perfectly integrated with some pre-existing systems, such as other car sharing and bike sharing systems, through their exposed APIs.

2.2 User Characteristics

As specified in the "Actors" section above, there is no relevant distinction that we have to make between our expected end users, simply because there are no particular kinds of users we are expecting use the application, once it is ready.
Users don't need any specific knowledge to make a good use of Travlendar+, and, moreover, users' main life occupations are not really relevant because our application can be used to schedule every kind of appointment, from business meetings to a football match with friends.

2.3 Assumptions, Dependencies and Constraints

2.3.1 Assumptions

- **[A1] Ubiquity:** Users cannot schedule two different activities at the same time of the same day. This means that the second activity must be scheduled on a different time of the day, or the first one must be deleted.
- **[A2] GPS precision:** to suggest the best mobility option, Travlendar+ has to acquire the position of the user though GPS, if the user hasn't inserted a starting point manually. We assume that those data have a maximum error of 10 meters.
- **[A3] Email:** Users' declared emails are supposed to be currently in use. When a user registers, his email is verified. From that

moment, then, we assume that the user will not stop using and regularly checking that mailbox.

- **[A6] Accidents:** Accidents and delays are frequent but most of the times online systems that calculate shortest mobility options get to know about them rapidly and take them into account for the solutions they propose. We assume that, if a user is following one of the mobility options given from Travlendar+ (see [A7]), unpredictable accidents that may cause delays have a rate of 10%. This means that the rest of the times (90%) users will be on time at their appointments.
- **[A7] Mobility options:** Users will follow the mobility options proposed by Travlendar+.
- **[A8] Refund policy**: The purchase of a ticket cannot be withdrawn.
- **[A9] Internet**: Users have access to the Internet.
- **[A10] Third part reliability**: The Third part systems we rely on will provide us the service we need at least 99,9% of the time.
- **[A11] Agreements with third part companies**: We assume that the agreements with the third part companies we will refer to have already been made and accepted by both parts.
- **[A12] Deployment in a restricted area:** As this is the first version, the application has been developed in order to be launched in a restricted area (Milan). A future expansion will be taken into consideration if the application is successful.
- **[A13] Tickets and Strikes:** It will be possible to implement the possibility to buy transportation tickets directly on the application only for the public transport systems that have public APIs that we can use. For the same transport systems it'll be possible to take into account strikes in the process of finding the best mobility option.

2.3.2 Constraints

- **[C1]** Confidential data inserted by the users must be stored in a secure way, according to the local actual privacy laws.
- **[C2]** Users' GPS position can't be acquired if the user himself didn't give his consent.
- **[C3]** Failures can happen, but when they happen the system must be able to restart from its status before the failure.
- **[C4]** Users' calendars must be private (e.g. not visible to any other user).

3.0 Specific Requirements

3.1 Functional Requirements

- **[R1]** Allow the users to manage already existing activities.
- **[R2]** Users should be able to log in to Travlendar+.
- **[R3]** Users should be able to register to Travlendar+.
- **[R4]** Users should be able to change their password whether they forget it.
- **[R5]** Users should be able to schedule new activities.
- **[R6]** Users should be able to set their own preferences that will be taken into account and will be applied to schedules every time this is possible and reasonable.
- **[R6.1]** *Specification*: The user can also set flexible activities (e.g. flexible lunch) , and, in particular, the modality "minimize carbon footprint" will be present.
- **[R7]** When necessary, users should be supported in buying public transports tickets directly on Travlendar+ and/or redirected on the correct external page.
- **[R8]** Users should be warned when they're scheduling an activity that is not physically possible due to a lack of time or that overlaps with other activities.
- **[R9]** Mobility solutions involving car and bike sharing systems must be taken into account, when possible, and proposed to the user when they represent the optimal solution.
- **[R10]** Users should receive a notification (e.g. email, push notifications) a little before the time they have to leave to go to the next appointment.
- **[R11]** The application should identify the best mobility option. Moreover, this should be done by appointment and by day (e.g., the app should suggest that you leave your home via car in the morning because meetings during the day will not be doable via public transportation).

3.2 Non-Functional Requirements

- **[NFR1]** After a user is logged in, he should be able to reach every functionality in less than 3 taps/clicks.
- **[NFR2]** The mobile application , when it will be developed, should work properly on Android, iOS (the most widely used) and, optionally, on Windows Phone.
- **[NFR3]** The desktop application should work at least on Windows 7 or higher and on MacOS X or higher.
- **[NFR4]** The system should be available at least 99,9% of the time over a year.
- **[NFR5]** The website should work fine at least on the best-known browsers (Safari, Google Chrome, Mozilla FireFox, Internet Edge).

3.3 Requirements and Assumptions Mapping

To prove the completeness of the requirements we provided and assumptions we made, we have to prove that when they're respected and verified, the goals are reached as a logical consequence.
The goals have been repeated below for a matter of simplicity.

- *[G1] Users should be able to use the system properly.*
  This goal can be reached thanks to functional requirements [R1], [R2], [R3], [R4] and assumptions [A3], [A9], [A10].

- *[G2] Allow users to schedule an activity.*
  This goal can be reached thanks to functional requirements [R5], [R1], [R8], and assumption [A1].

- *[G3] Allow users to set constraints and preferences.*
  This goal can be reached thanks to functional requirements [R6], [R6.1].

- *[G4] Simplify procedures and reduce the number of operations that a user has to do to schedule an new activity (in general).*
  This goal can be reached thanks to functional requirements [R1], [R5], [R6], [R6.1], [R7], [R8], [R9], [R10], [R11], non-functional requirement [NFR1].

- *[G5] Make sure users are on time at their scheduled appointments.*
  This goal can be reached thanks to functional requirements [R7], [R8], [R9], [R10], [R11] and assumptions [A2], [A6], [A7], [A9], [A10], [A11].

3.4 Reasons that justify some requirements

For the sake of clarity, the less obvious requirements are explained better here:

- [NFR1] – Many applications have a complicated structure that makes it hard or disagreeable to reach a particular functionality. We went to keep it the simplest we can, so that our application will result easy to use and this will not constitute a reason for our users to stop using it. In addition, if it's easy to use, users will find it easier to exploit all its functionalities and get the best from our application.
- [R9] – Though this can be not the easiest thing to implement, we want to consider all the possibilities to advice the user with the best mobility option.

3.5 External Interfaces Requirements

3.5.1 Software Interfaces

The system will have to rely on a third part map and geo localization service in order to provide users with more precise and customized notifications and directions. To furnish Travlendar+ users with a high-quality service Google Maps (https://developers.google.com/maps/) and Google Geolocation (link to: Google Geolocation API) APIs will be used. Additionally, in order to store users' login credentials, calendar's data and preferences, the application will make use of a commercial DBMS (Database Management System). For this purpose MySQL (Version: 5.7.19, available at https://www.mysql.com) has been chosen. Moreover, to provide some functionalities Travlendar+ has to interface with an external web server (NginX 1.12.2).
Furthermore, the system needs to interface with some third part apps and services:

- a weather forecast service in order to provide more advanced and accurate suggestions and instructions. Yahoo! Weather APIs will be used (https://developer.yahoo.com/weather/).
- Third part bike sharing apps and websites such as Ofo and Mobike, for the Milan Metropolitan Area, to be integrated in the application.
- Car sharing services such as Enjoy and Car2Go APIs to integrate this kind of function in Travlendar+.
- Third part local public transportation apps and website for ticket purchasing.
- Uber APIs in order to integrate this kind of service in Travlendar+.

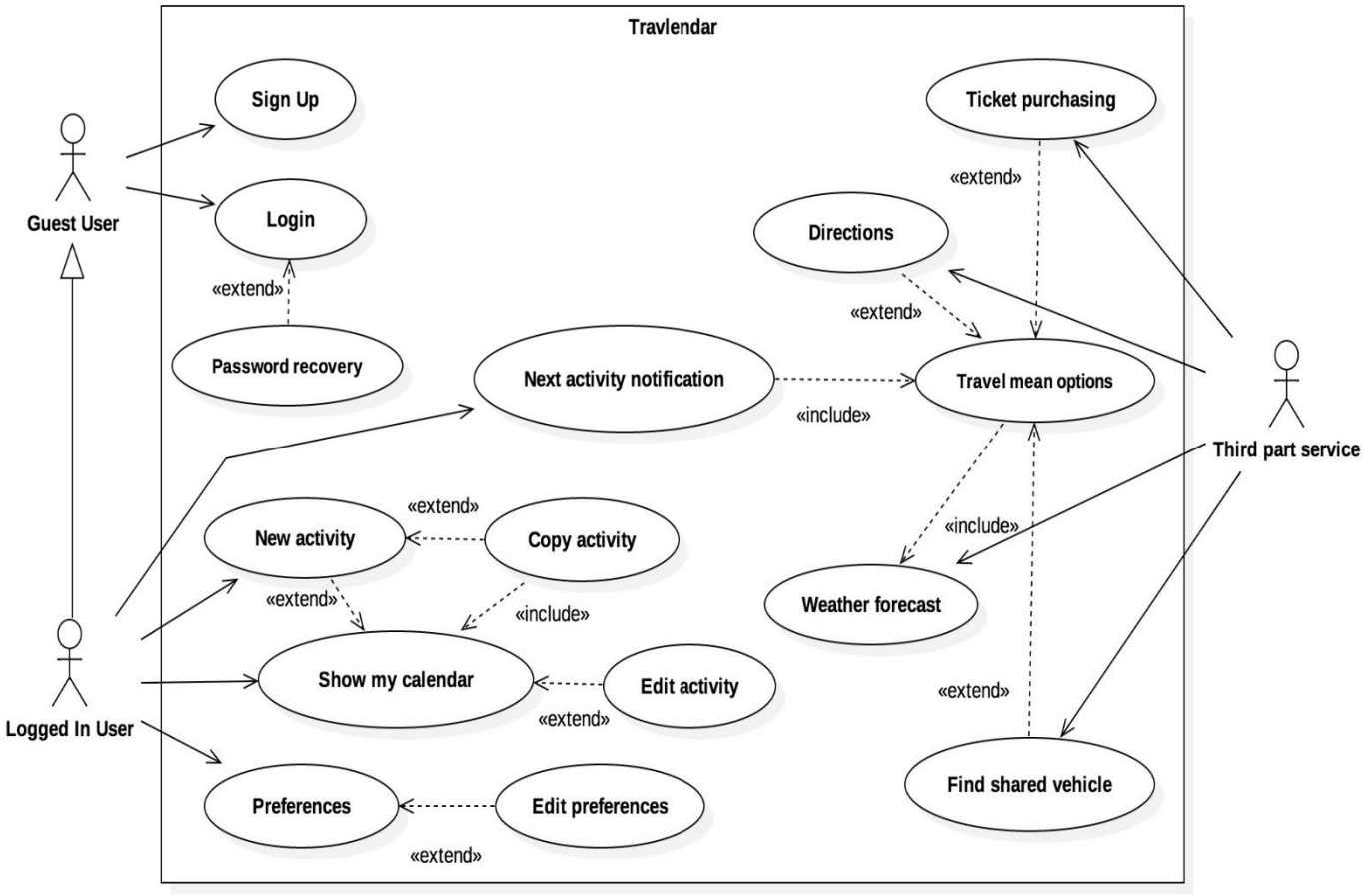
Furthermore, the Travlendar+ application has been thought and will be developed in order to work on almost any mobile device (see [NFR2]) and with all best-known web browsers (supported web browsers (see [NFR5]).

3.5.2 Communication Interfaces

The entire system will make use of TCP-IP protocol for Internet communication with the web server, the DBMS and users. The website will also make use of HTTPS for security purposes and of SMTP protocol to manage its mail component (e.g. send confirmation mails and handle the password recovery routine). No IMAP or POP3 components are required since the initial version of the system has no need to receive emails.

3.6 UML diagrams and charts

3.6.1 Use Cases Diagrams and tables



The use case diagram above presents the use cases in the Travlendar+ application. However, in the following tables just few of them are described in detail since they are considered the most relevant ones w.r.t. final user's interactions with the application.

In the tables, each use case is analysed presenting the actors involved, entry and exit condition(s) for the use case, the regular flow of events and exceptions interrupting it.

We would like to highlight the fact that the diagram, and consequently the Travlendar+ application, is easily extensible. For instance, a "Premium" user can be added if the application will ever be upgraded, in this case just adding a new generalization level to the user's hierarchy.

[UC1] Login

Name	Login
Actors	Guest User
Entry Condition	The user is already registered and just needs to log in the application.
Flow of events	- The user launches either the mobile or desktop version of the application - The application homepage is displayed (for further information see [UI1]) - The user must provide his username and password and click on the proper button - The application queries the DB to validate credentials and gives a response to the user
Exit condition	The user successfully gains the access to the application.
Exceptions	If either the password provided is not correct or the username does not exist, the system notifies the mistake.

See [BPMN1] for a visual representation.

[UC2] Sign Up

Name	Sign up
Actors	Guest User
Entry Condition	The user is not yet registered, so he should proceed with the registration routine in order to use Travlendar+ services.
Flow of events	- The user launches either the mobile or desktop version of the application - The application login page is displayed (for further information see [UI1]) - The user must choose the sign-up option, clicking on the proper button - The system shows the sign-up form asking for the mandatory information that is: username, password, email or Facebook/Google account - Data are sent and stored in the DB
Exit condition	The user successfully completes the registration routine and gains the access to the application
Exceptions	Moreover, if any mandatory information to complete the registration is incorrect or missing, an error screen is displayed.

See [BPMN4] for a visual representation.

[UC3] Password Recovery

Name	Password Recovery
Actors	Guest User
Entry Condition	The user is registered and needs to use the password recovery routine.
Flow of events	- The user launches either the mobile or desktop version of the application - The application homepage is displayed (for further information see [UI1]) - The user clicks the password recovery button - The user inserts his username - If the username is correct, the system starts the password recovery routine
Exit condition	An e-mail message is sent at the e-mail address provided during the registration with an attached link to recover the password.
Exceptions	-The link attached to the email expires and the process is interrupted.

See [BPMN2] for a visual representation.

[UC4] New fixed activity

Name	New activity
Actors	Logged-In User
Entry Condition	The user is logged in and the application is launched.
Flow of events	- The user clicks on the proper button (for further information see [UI8]) - The application starts the routine and, according to user's preferences, it asks for activity date, activity location, activity label, guests and notes about the activity, tag. - The system shows the activity added in the calendar and asks for user's confirmation - Data are sent and stored in the DB
Exit condition	A message is displayed to notify the user that he successfully added a new activity to his calendar.
Exceptions	Ether the activity cannot be added due to date, time (consistency with other activities) issues or the user interrupts the routine.

See [BPMN5] for a visual representation.

[UC5] Edit activity

Name	Edit activity
Actors	Logged-In User
Entry Condition	The user is logged in and the application is launched.
Flow of events	- The user selects an activity between those in his calendar - He clicks the proper 'edit' button (for further information see [UI4]) - The system starts the routine allowing the user to edit different data fields of the activity - The user clicks the 'save' button as he finishes to edit the activity - The system saves changes in its server /DB
Exit condition	A notification makes the user aware that changes performed have been saved.
Exceptions	Changes performed makes the activity inconsistent with other activities (so user cannot save his changes).

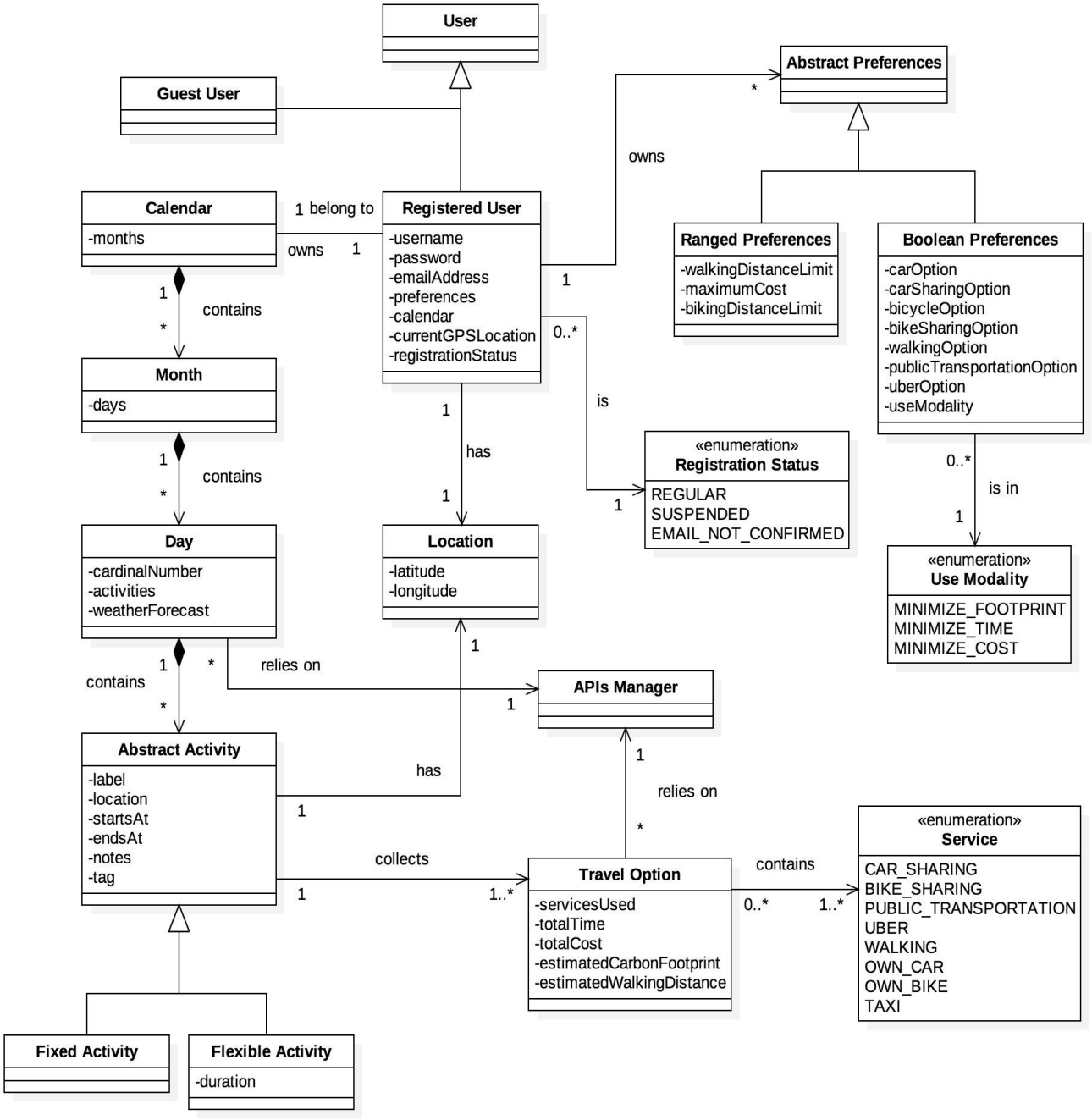
See [BPMN3] for a visual representation.

[UC6] Edit preferences

Name	Edit preferences
Actors	Logged-In User
Entry Condition	The user is logged in and the application is launched.
Flow of events	- The user clicks on the proper button (for further information see [UI2]) - User's profile and his preferences are displayed - User can perform changes on preferences about favorite travel means and distances, third part services to be used - If changes are performed, the system saves them in its server
Exit condition	The system automatically saves the changes.
Exceptions	None.

See [BPMN6] for a visual representation.

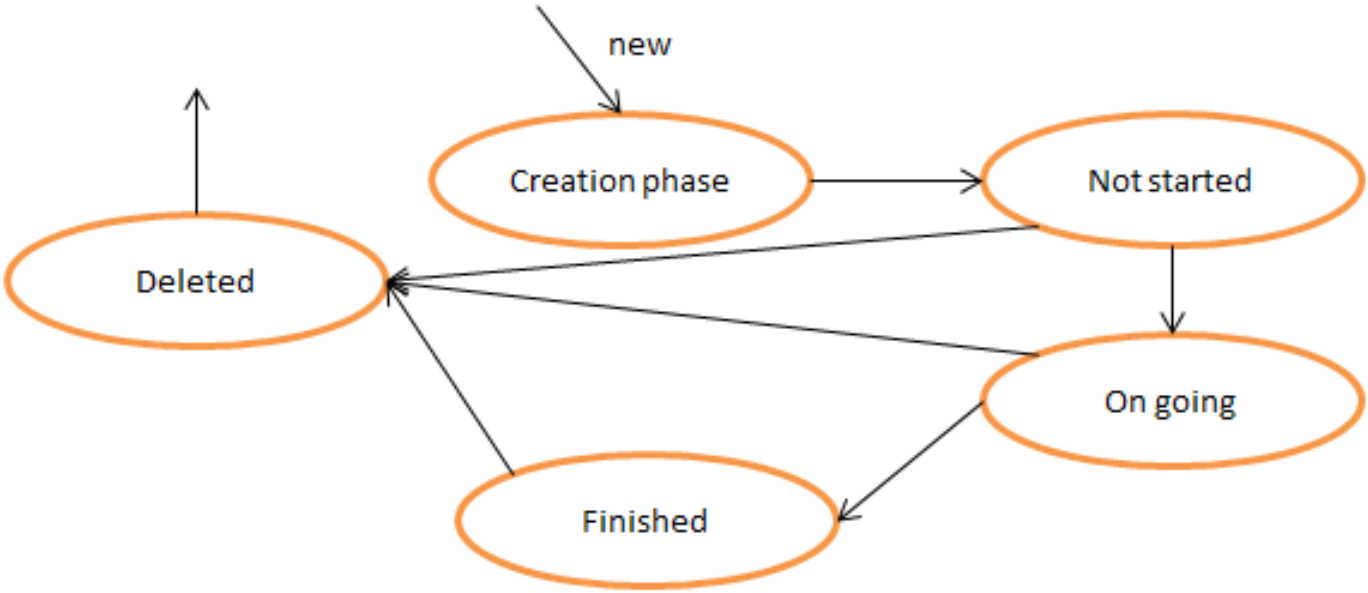
3.6.2 Class Diagram



The class diagram above contains classes and relations regarding the application functionalities to provide to the user. Network and other architectural design classes are missing in this diagram, the purpose of which is to present entities involved in the main user's interactions with Travlendar+ for the sake of clarity.

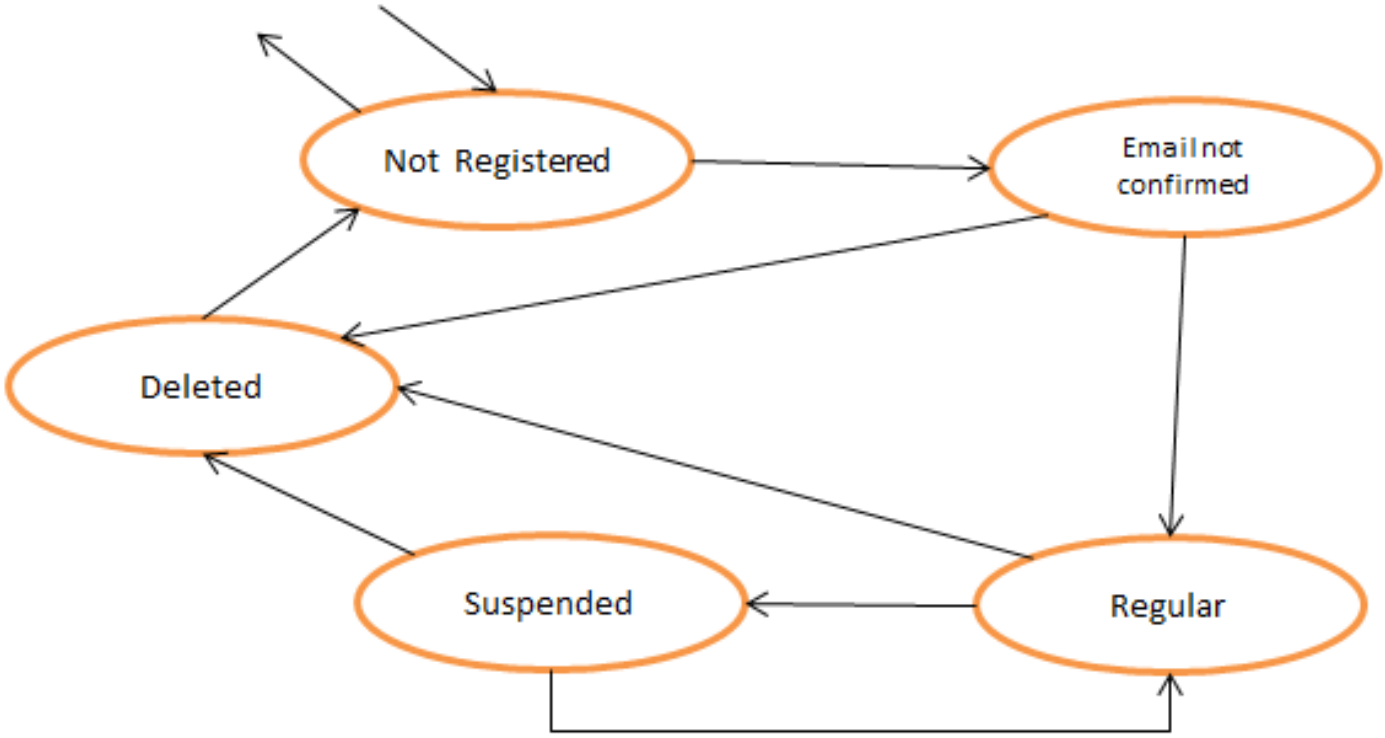
3.6.3 State chart Diagrams

- **Event lifecycle:** The lifecycle of an event can be modeled as follows:



The cycle starts with the creation phase. After the event is scheduled its status is “Not started”. The evolution through the “On going” and “Finished” states is obvious and automatic. Events in the “Finished” state are deleted automatically after a certain amount of time. From the “Not started” and the “On going” state, the event can be deleted by the user. The “Deleted” state is final: once an activity is in that status, it can’t be restored and the system has already deleted any data regarding it permanently.

- **Registration lifecycle:** The lifecycle of an user's account can be modeled as follows:

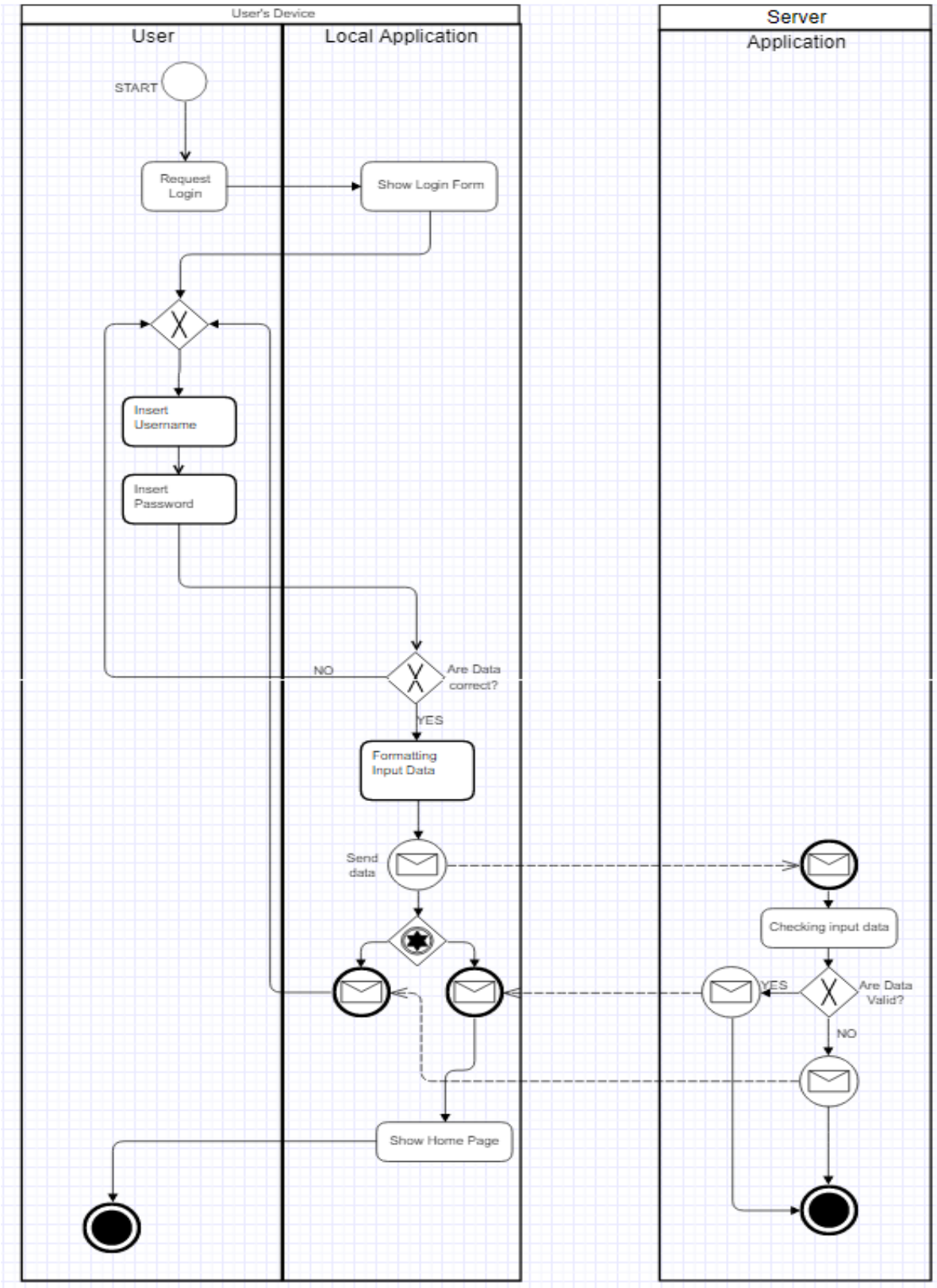


The "Not Registered" state is the initial state and final state. In that state the user doesn't have an account and has not even started the registration process. So, as a consequence, the system has no data about the user. When a user creates an account, its registration state becomes "Email not confirmed" and when he completes the registration by following instructions received by email the state becomes "Regular".

From that moment the user has access to all the functionalities offered by Travlendar+. From the state "Regular", a user can decide to suspend or delete his account, changing its state respectively to "Suspended" or "Deleted". While the "Regular" status can be restored from the "Suspended" status whenever the user wants, the "Deleted" one is irreversible, and after a certain amount of time, a Deleted account is completely erased from the system and the user's status becomes "Not Registered" again. If the application will ever include any pay-to-use features, a "Premium" status can be added quite easily to the chart.

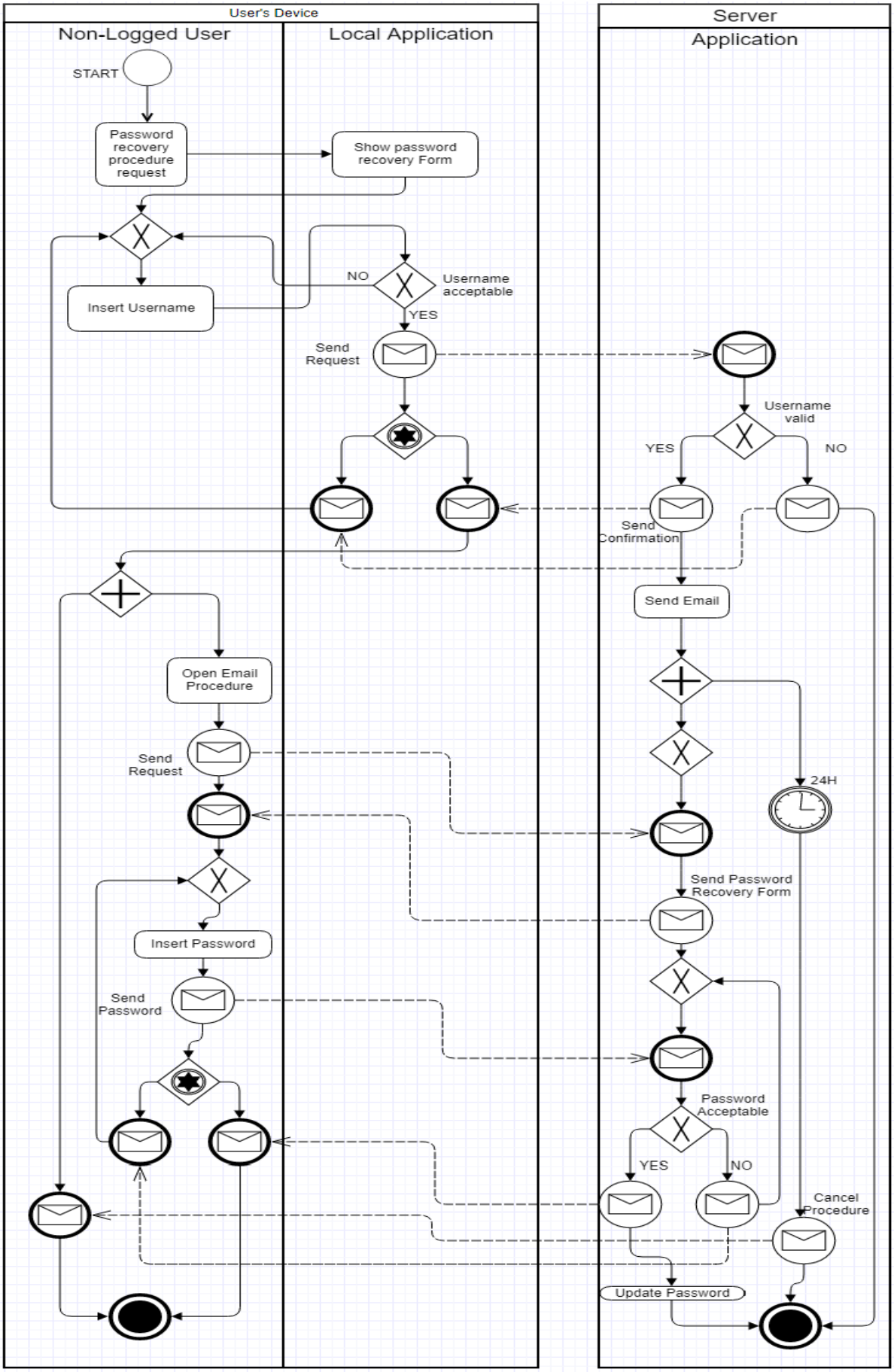
3.6.4 BPMN Diagrams

[BPMN1] Login procedure



The BPMN diagram showed above represents in a relatively simple way the procedure that a user has to follow to log into Travlendar+. The implementation of the procedure with this events flow is compliant with the presence of [R2].
The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.
The process starts when the user requests it and the operations needed to show the login page and ask him his username and password are handled out locally, through the application that the user has previously installed on his device. Then, if the inserted data are acceptable (e.g. fields have not been left empty and they contain only accepted characters), the request is sent to the server through the communication interface. The answer (e.g. login successful/ denied) is sent back to the user's device and the process can end if the answer is positive or restart otherwise.
The process terminates on the server as well.

[BPMN2] Password Recovery Procedure

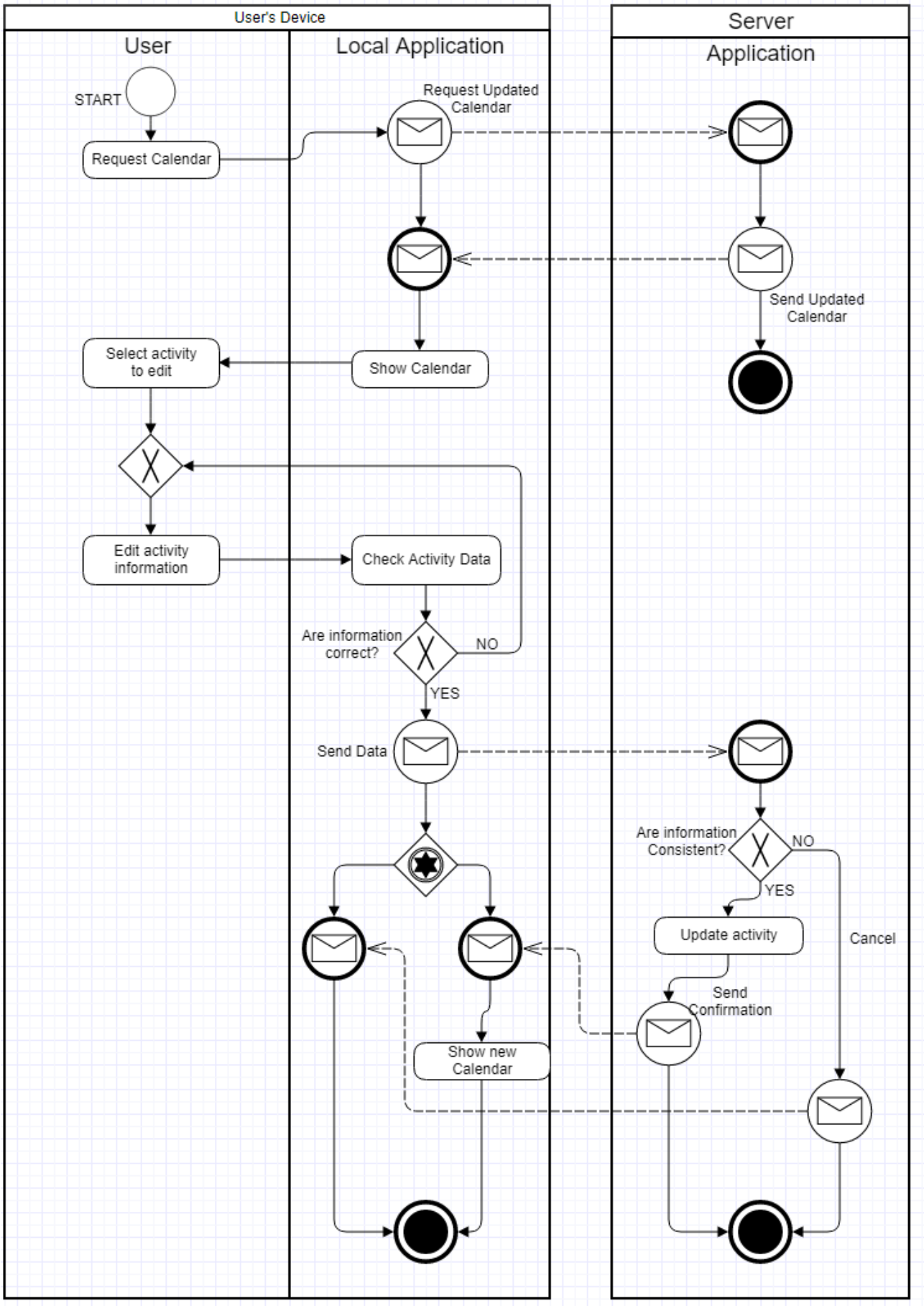


The BPMN diagram showed above ([BPMN2]) describes the interaction required for a user to activate the password recovery procedure. The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.
The process starts when the user requests the local password recovery form. The completion of that form with a valid username, where the validity is checked both locally and remotely, will cause the system to send the user an Email containing the customized instructions that allow the user to change is password.

The 24H timer in the diagram that starts at this point models the fact that the customized URLs and instructions that have been sent to the user have a validity of 24 hours. After that time those can be considered expired and the user has to start a new process to change his password.

When a user follows the whole procedure and sets a new password, the system updates it and the procedure terminates.

[BPMN3] Edit an Activity

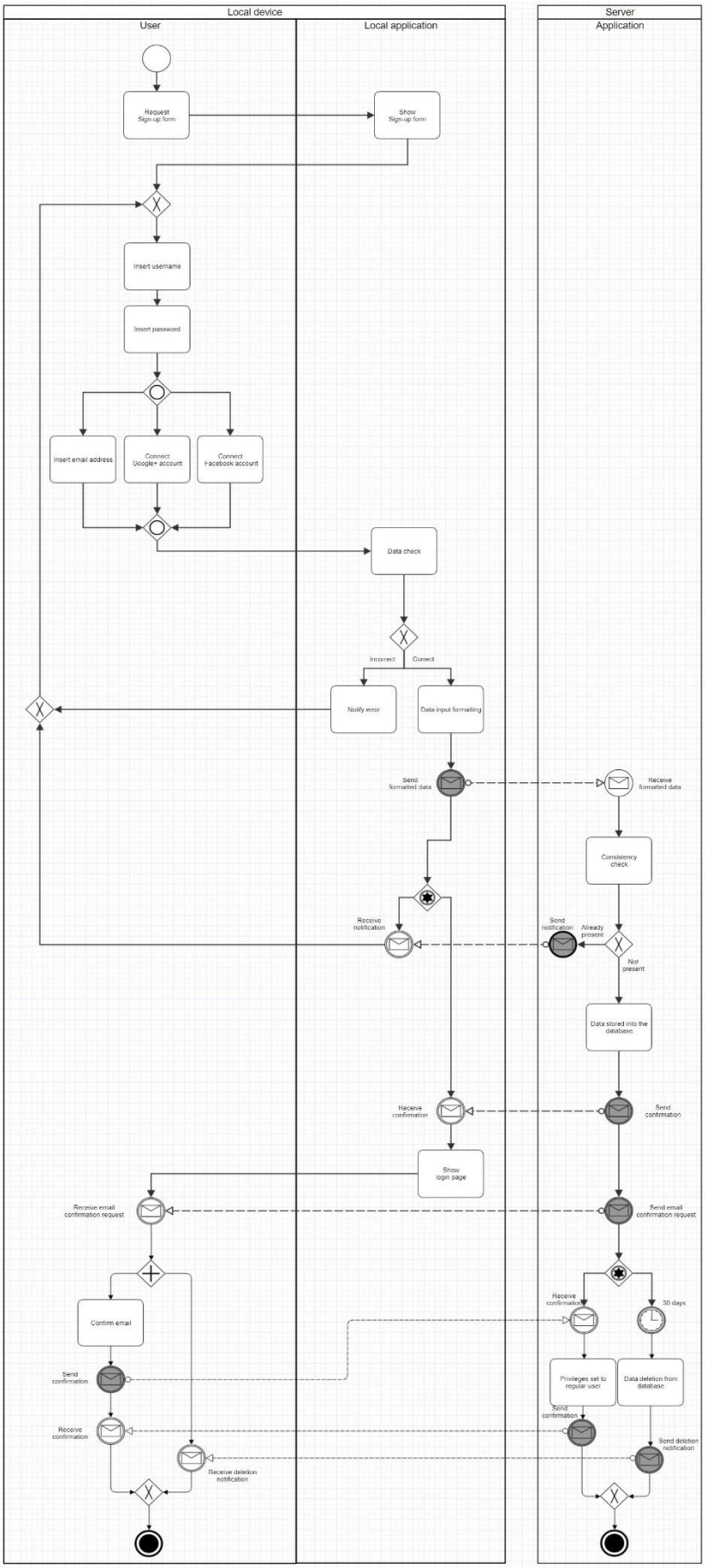


The BPMN diagram showed above ([BPMN3]) describes the interaction required for a user to edit an already existing activity. The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.
The reach the functionality, the user has to request the calendar and select the activity. Though those operations are not directly involved in this process, they have been represented anyway to clarify that the calendar is updated locally before it is showed to the user. This happens because we want to avoid inconsistencies between the local calendar and data stored in the server. When the user inserts the new activity data, locally it's checked if they're correct and don't contain unaccepted characters, and remotely it's checked if they're consistent with the present calendar. The remote check is necessary because this sequence of actions may cause inconsistencies:

- The user requests the calendar on a device (device1);
- The user requests the calendar on a second device (device2);
- The user modifies an activity (a1) on device1;
- The user modifies another activity (a2) on device 2. If a2 overlaps with the new a1, the local check is passed but the remote check allows to avoid inconsistencies.

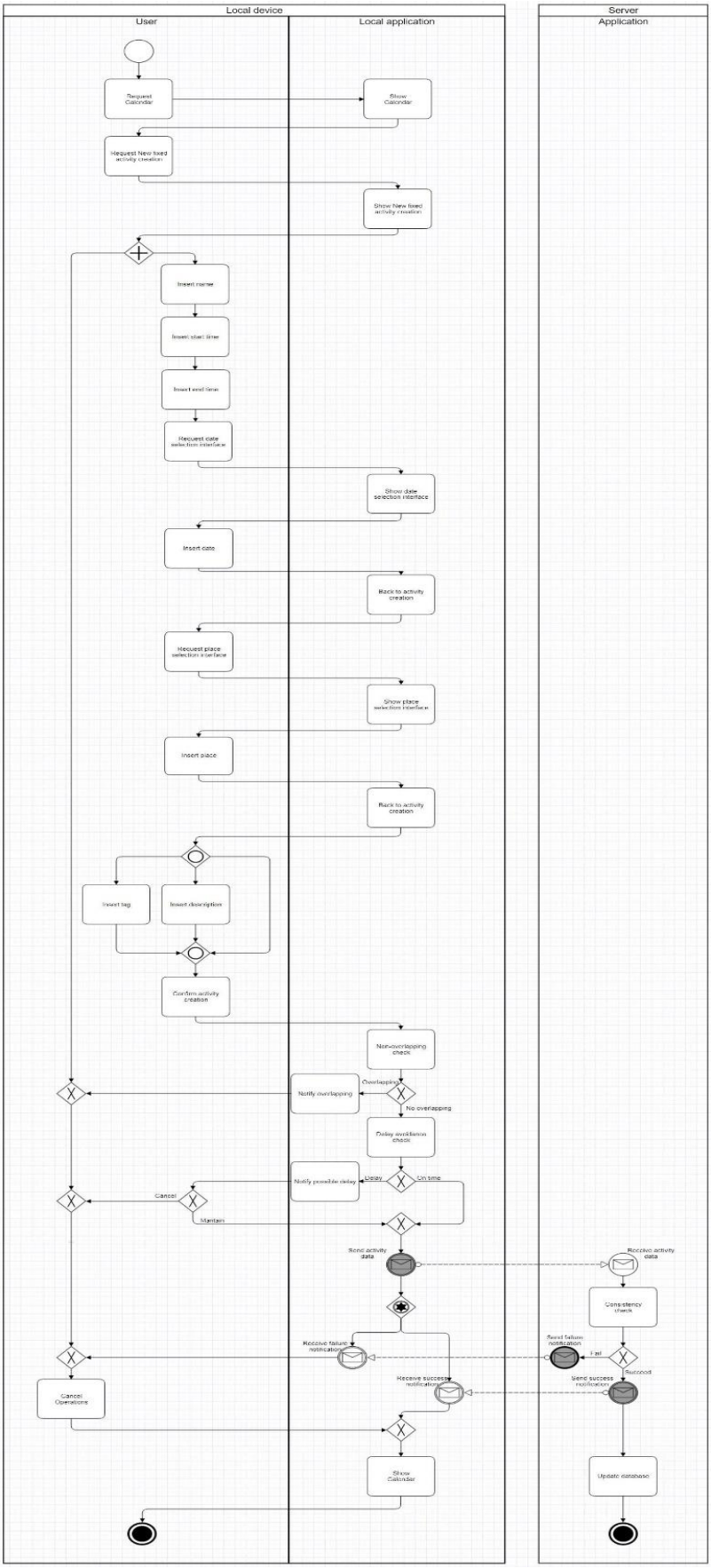
If there are no inconsistencies and the modified activity is accepted, the calendar is updated. If the modified activity is not accepted, a message is sent to the user. In both cases the process terminates on both client-side and server-side.

[BPMN4] Register



The BPMN diagram showed above ([BPMN4]) describes the interaction required for a user to register to the application. The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.
Through the given register form, the user is asked to insert his username and password, as well as at least one option between email address, Google+ account and Facebook account.
After a data check to make sure that the data is consistent (e.g., there are no forbidden characters and the contact information provided is not fake), the data is sent to the server, that stores it into the database.
At the end, the login page is shown to the user.

[BPMN5] Create a fixed activity

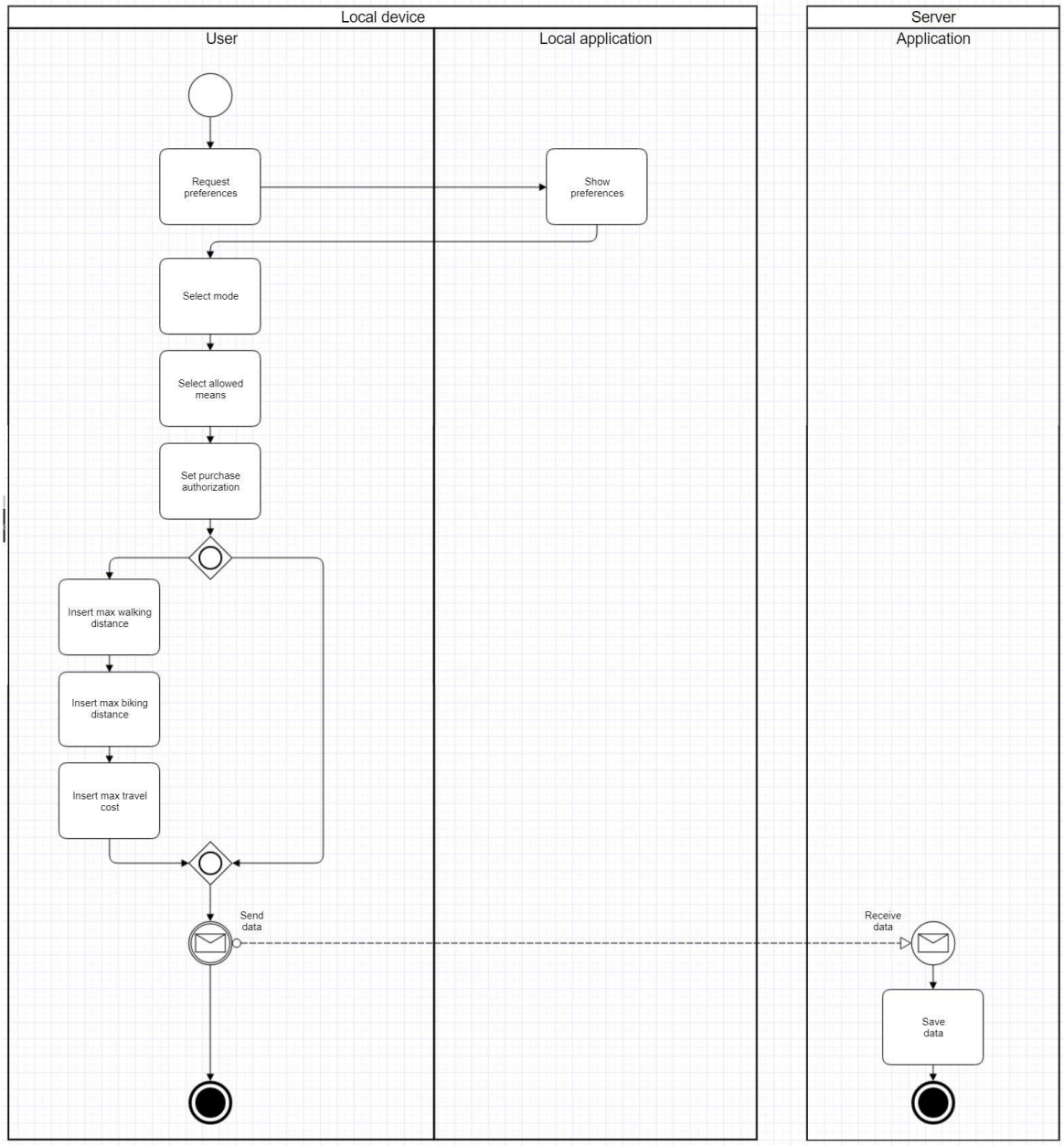


The BPMN diagram showed above ([BPMN5]) describes the interaction required for a user to create a new activity. The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.

Through the new activity creation interface, the users can set different parameters to customize their activity. Some of them require the use of another interface, like the date and place selection, while some other are optional, like tags and description. After confirmation, the activity creation is processed and two main checks are carried out: the non-overlapping check, which makes sure that the new activity does not overlap with any existing activity, and the delay avoidance check, to make sure that the time between the new activity and the previous one is sufficient to reach the location where the new activity will take place. While a failure of the first check will cause the activity to be cancelled, in case the delay avoidance check fails, the user can still decide to maintain the activity.

Finally, the data about the activity is sent to the database and the calendar interface is shown to the user.

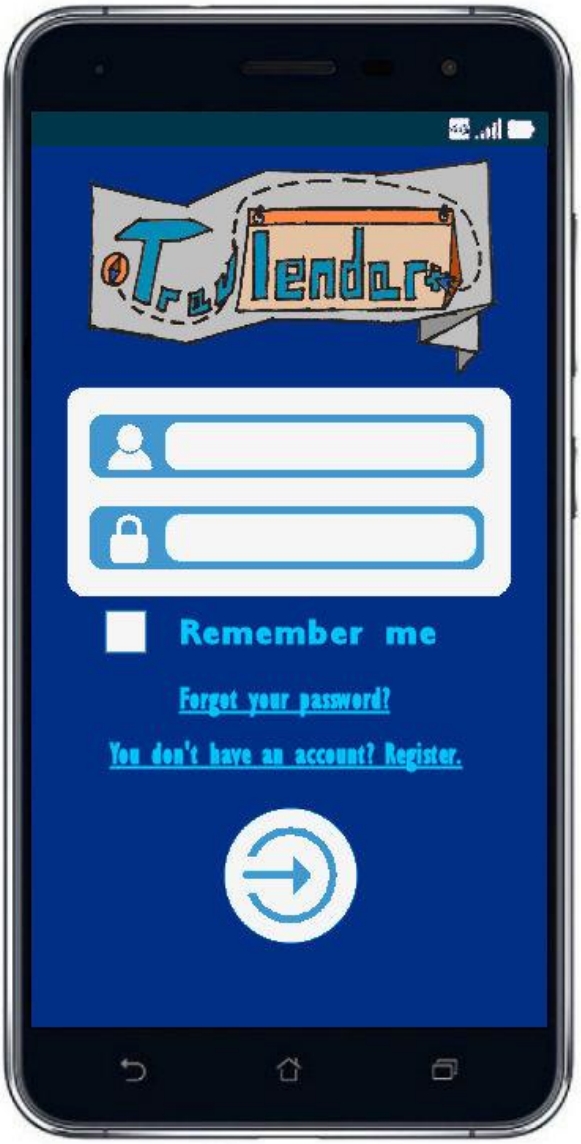
[BPMN6] Preferences selection



The BPMN diagram showed above ([BPMN6]) describes the interaction required for a user to set his preferences. The diagram is divided in 2 pools and 3 lanes in total. While the first pool represents the client side, basically the user and his device, the second pool represents the Server where data about users, preferences and calendars are stored.
Both the use of pre-defined options and the insertion of personal constraints, like the maximum walking distance, are featured, though the definition of the latter is not compulsory.

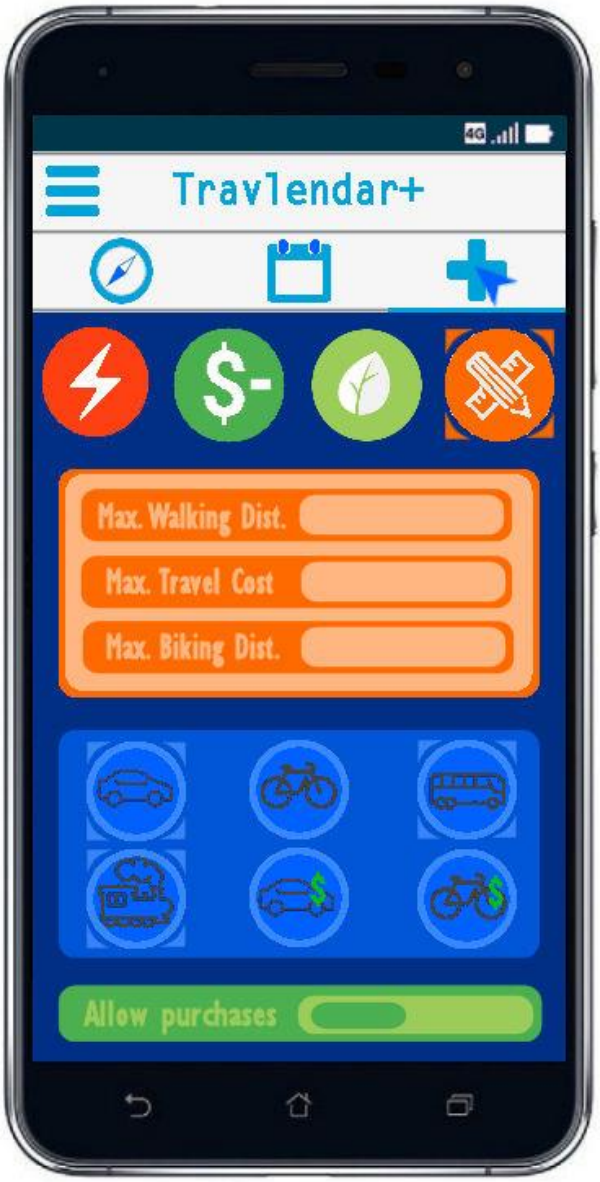
3.7 Graphical User Interface Examples

- **[UI1] Login**



Users can log in by filling in the fields with their data. If they don't have an account, they can register. If they can't remember their password, they can start the recovery procedure. Before logging in, by checking the box under the password field, users can save their personal data to make future accesses quicker.

- **[UI2] Preferences Settings**

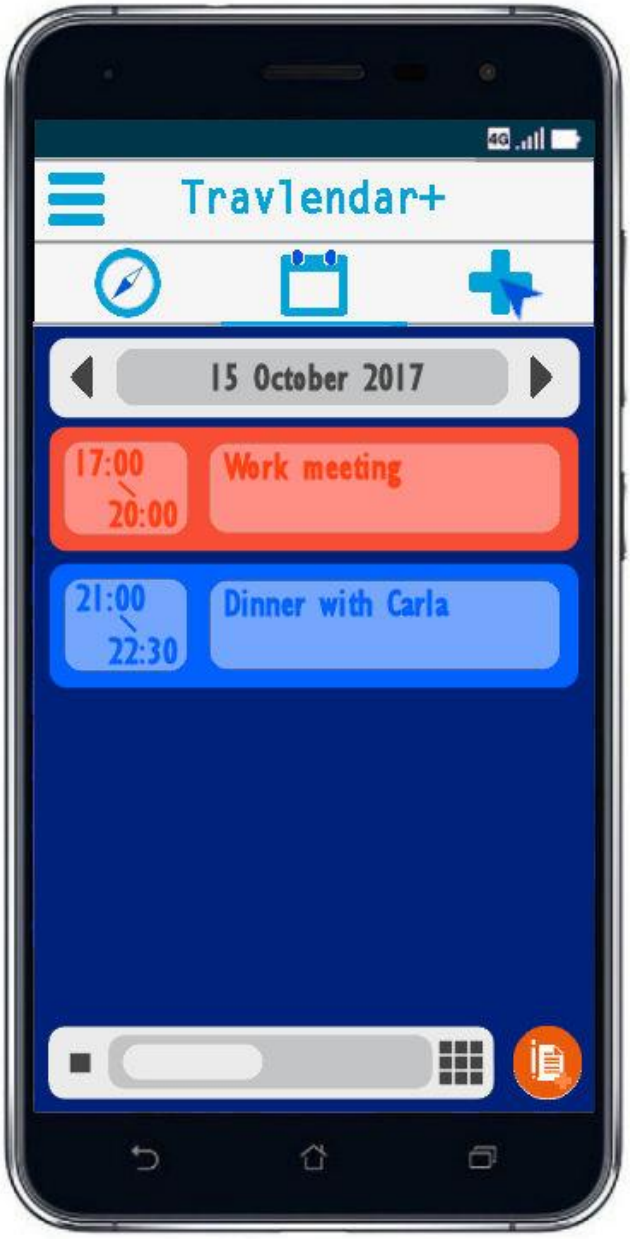


Users can set their preferences for their travels. The applications has three predesigned modes, which are fastest itinerary mode, cheaper itinerary mode and ecological mode. The application also offers a custom mode, in which the user can choose the maximum walking distance, biking distance and travel cost. Underneath, the user can choose which travel means are allowed to be taken in consideration by the applications. The choices are personal car, personal bicycle, public urban means, trains, taxis/car sharing services and bike sharing services. Finally, users can authorize the application to automatically purchase tickets if necessary for the itinerary.

- **[UI3] Calendar (month)**

Users can use this month-view to have a general idea of their activities. The current week is automatically selected, but any other week can be selected for further information. By clicking on the specific day afterwards, a quick view of the activities, shown by tag, is visible underneath. Users can also change the selected month, change view or create an activity.

[UI4] Calendar (day)



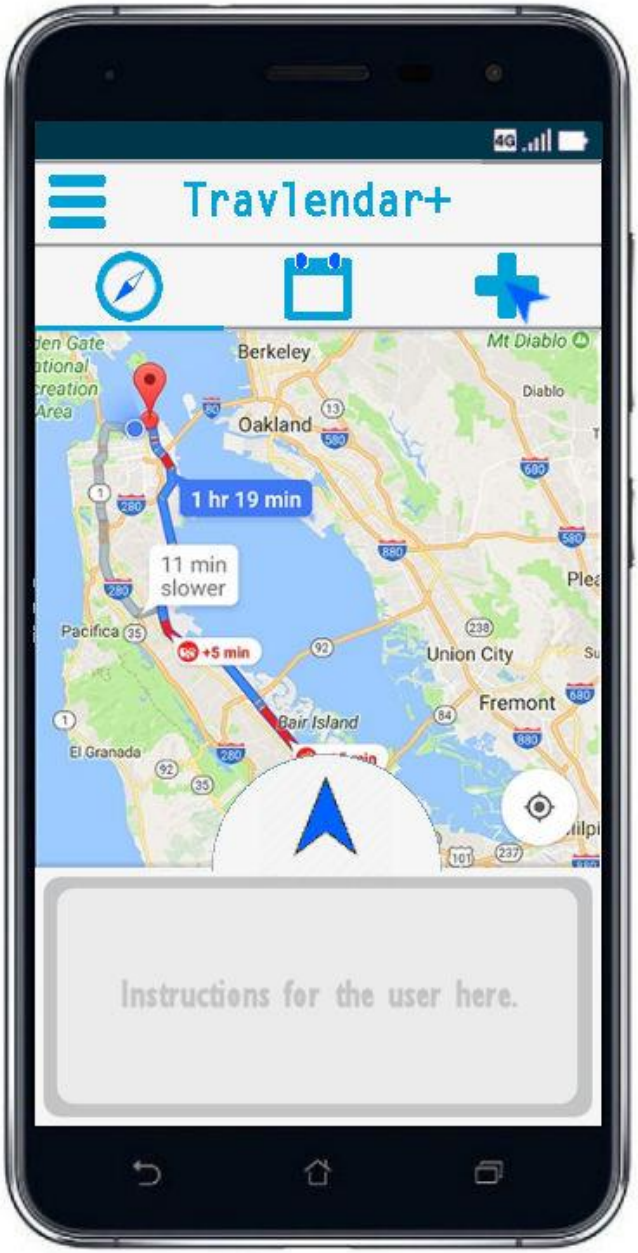
This is the home page. From here, users can see their activities for the day. By clicking on each of them, they can have more information or just perform actions on them, like copy or delete. The view can also be changed to month-view through the button at the bottom of the screen. A new activity for the day can be created by pressing the orange button.

The bar on top is always visible. Users can use it to access the three main functions of Travlendar+: the map, the calendar or the user preferences. A menu is also avaiable to show the main information about the user.

- **[UI5] Top-left corner menu**

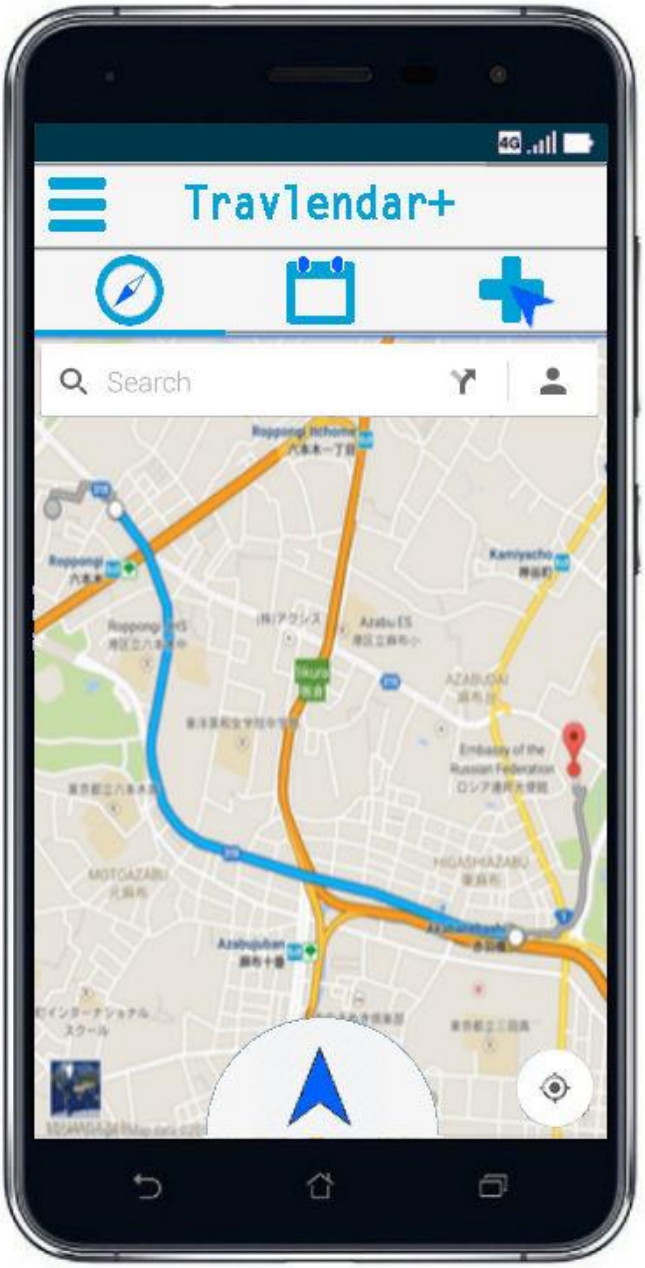
From the menu, users can see their username, access their account information, which include credit card information and ticket/passes associated to the user, select their saved address - so that they have a quicker way to select places -, send us feedback, see the Frequently Asked Questions or just log out.

- **[UI6] Travel Options**



Users can choose the daily itineraries and see the instructions to reach the place where the next activity will take place in the specified box.

- **[UI7] Travelling phase**



Users can look for a specific place on the map. When selecting a place, they can decide to create an activity associated to that place. By pressing the arrow at the bottom, the application will compute several daily itineraries and will show them to the user, so that he can choose the one he prefers.

- **[UI8] Fixed Activity Creation**

Users can insert the name of the activity, the starting and ending time, optionally the description and a tag, like "work" or "free time", each associated to a unique colour. To associate the activity to a day, users are brought to the calendar interface, from which they can select a day. If the new activity creation is selected from the day-view calendar interface, the day will be selected automatically. To associate the activity to a place, users are brought to the travel interface, from which they can select a specific place. They can also select it from their saved addresses. If the new activity creation is selected from the travel interface after selecting a place, that place will be selected automatically. At any moment, the user can either confirm the creation or cancel operations.

3.8 Software System Attributes

3.8.1 Reliability

At most two failures per year are allowed in system's hardware, i.e. in the web and the database server used to provided Travlendar+ functionalities.

3.8.2 Availability

As mentioned in NFR4, the entire system should be available at least 99.99% of the time over a year, so it could go down (i.e. be unreachable and unusable) for less than an hour overall during a year. For this purpose, the server side of the system should be highly scalable, in order to face a great number of connections/users, strong against software and hardware failures (see reliability above) and possibly mirrored over several databases and servers, so that users never face uselessness of services.

3.8.3 Security

Since the system manages users' personal and sensitive data, it must guarantee the four primary security requirements: integrity, confidentiality, authenticity and authentication. For this reason, it should be strong against most common external attacks which are sniffing attacks, DDoS (Distributed Denial of Service), SQL Injection attacks, man-in-the-middle attacks and computer viruses and worms.

In order to guarantee integrity and authenticity both passwords and server's disks will be encrypted. To ensure confidentiality and authenticity web communications will be encrypted using HTTPS protocol. The system will guarantee also that only registered, authenticated users will have access to the system (for further information about privacy and personal data handling see the References section).

Furthermore, the system should be tough enough against natural misfortunes (such as fire, floods, earthquakes etc.), so adequate prevention will be taken into account. All user's personal data such as first name, last name, email address and external accounts information etc., provided during the registration to Travlendar+, must be stored and handled in

respect of the local government laws about privacy and personal data handling. In Italy this policies are contained in D.Lgs. n. 196/2003.

3.8.4 Portability

As described by NFR2 and NFR3, the Travlendar+ application should be platform independent, so a cross platform programming language will be used, for instance Java. This choice takes also into account new mobile and desktop OSs that could be developed in the future.
In addition, our application has no particular hardware requirements and this increases portability.

4. External References

4.1 External documentation

- Privacy Law (Italian): D.Lgs. n. 196/2003.
- API Yahoo! Weather: link.
- API Google Maps: link.
- API Google Geolocation: link.
- MySQL documentation: link.
- NginX 1.12.2 documentation: link.

4.2 Used Tools

- BPMN Modeler Gliffy: link.
- Corel PaintShop Pro 2018: link.
- StarUML (for the Class Diagram abd Use Case Diagram) : link.
- Alloy Analyzer: link.
- GitHub: link.
- GitHub Desktop: link.

5. Alloy

5.1 Code

```
// MATTEO BIASIELLI, EMILIO CAPO, MATTIA DI FATTA
/* INITIAL COMMENT */
/* This model is based on the UML class diagram showed in the RASD file.
Our intention is to represent the system with the imposed constrants,
described in the above mentioned file, and to show that it is consistent and
that admits some possible worlds which can represent real situations.
*/

/*DIFFERENCES WITH THE UML DIAGRAM
One of the main differences with the UML diagram is that here we will not
model dates as a triple <day, month, year> for two reasons:
-it's not strictly necessary to show what we want to show;
-it's easier to use normal integers on Alloy.
In spite of this, we will model days as simple positive integers.
Activities' start and end time will be modeled as integers included in [0,24),
assuming that activities' duration can be just an integer number of hours.
*/

open util/integer

abstract sig RegistrationState{}
/*we represented the only useful registration states.
Not registered state and deleted state were not useful in our Alloy representation*/
one sig EMAIL_NOT_CONFIRMED extends RegistrationState{}
one sig REGULAR extends RegistrationState{}
one sig SUSPENDED extends RegistrationState{}

/*This represents the actual day and time.
It is useful to determine and assign states to activities*/
one sig SystemTime{
   day: one Int,
   time:one Int
}{
   time>=0 && time<24 && day>=0
}

abstract sig EventState{}

/*As for the registration states, we represented all and
only the states that are useful to our representation*/
one sig ON_GOING extends EventState{}
one sig NOT_STARTED extends EventState{}
one sig TERMINATED extends EventState{}

/*Definition of Boolean*/
abstract sig Boolean{}
one sig TRUE extends Boolean{}
one sig FALSE extends Boolean{}
```

```
/*Definition of the possible preferences.
For now, only few of the simplest preferences are represented.
This Representation, anyway, can be obviously and easily extended to all kinds of transportation*/
sig Preferences_Set{
   carAvailable: one Boolean,
   bikeAvailable: one Boolean,
   maxWalkingTime: one Int,
   maxBikeTime: one Int,
   maxCarTime: one Int,
   maxTravelCost: one Int,
   maxPublicTransportTime: one Int
}{
   maxWalkingTime>0 && maxBikeTime>=0 && maxCarTime>=0
   && maxTravelCost>=0 && maxPublicTransportTime>=0
      &&
   maxWalkingTime<=10 && maxBikeTime<=10  && maxCarTime<=10
   && maxTravelCost<=10  && maxPublicTransportTime<=10
}

/*Representing an abstraction of the places*/
sig Place{}

/*Abstraction of Emails*/
sig Email{}

sig User{
   position: one Place,
   calendar: one Calendar,
   state: one RegistrationState,
   email: one Email,
   preferences: one Preferences_Set
}

sig Calendar{
   activities: set Activity,
   breaks: set FlexibleBreak
}


sig Activity{
   place: one Place,
   travel: one TravelOption,
   state: one EventState,
   startDay: one Int,
   endDay: one Int,
   startTime: one Int,
   endTime: one Int
}{
   /* The Activity start time must be prior to the end time.
   Same with the start day and end day*/

   startDay>=0 && endDay>=startDay &&
   (startTime<endTime || endDay>startDay) &&
   startTime>=0 && startTime<24 && endTime>=0 && endTime<24
}
```

```
/*Flexible break is assmed to start and finish on the same day for a matter of semplicity*/
sig FlexibleBreak{
   day: one Int,
   startTime: one Int,
   endTime: one Int,
   duration: one Int
}{
   /* The Activity start time must be prior to the end time.*/
   startTime<endTime &&
   startTime>=0 && startTime<24 && endTime>=0 && endTime<24 && duration>0
   && duration<=sub[endTime,startTime]
}

/*In this simplified model of a travel option, only times are represented.
We assumed that each unit of time on public transports has cost 1.
For a matter of simplicity, and to avoid total time to exceed integer bitwidth,
each value is bounded at 10 as max value*/
sig TravelOption{
   startPlace: one Place,
   endPlace: one Place,
   walkingTime: one Int,
   bikeTime: one Int,
   carTime: one Int,
   publicTransportTime: one Int,
   travelCost: one Int,
   totalTime: one Int
}{
   startPlace!=endPlace && walkingTime>=0 && bikeTime>=0 && carTime>=0 &&
   travelCost>=0 && totalTime>=0  && publicTransportTime>=0
      &&
   walkingTime<=10 && bikeTime<=10  && carTime<=10
   && publicTransportTime<=10
      &&
   travelCost=add[0,publicTransportTime] &&
   totalTime=add[walkingTime,add[bikeTime,add[carTime,publicTransportTime]]]
}


/*There is at least one travel option between each user's position and
its activities places*/
fact placesAreConnected{
   all u:User|( all a:u.calendar.activities|(some t:TravelOption| (t.startPlace=u.position
      && t.endPlace=a.place)))
}

/*Travel options chosen for each activity must satisfy user's preferences and must
connect the user's position to the activity's place*/
fact travelOptionsSatisfyPreferences{
   all u:User|all a:u.calendar.activities| (
      a.travel.startPlace=u.position && a.travel.endPlace=a.place &&
      a.travel.walkingTime<=u.preferences.maxWalkingTime &&
      a.travel.bikeTime<=u.preferences.maxBikeTime &&
      a.travel.carTime<=u.preferences.maxCarTime &&
      a.travel.publicTransportTime<=u.preferences.maxPublicTransportTime &&
      a.travel.travelCost<=u.preferences.maxTravelCost) &&
      (u.preferences.carAvailable=FALSE implies a.travel.carTime=0) &&
      (u.preferences.bikeAvailable=FALSE implies a.travel.bikeTime=0)
}
```

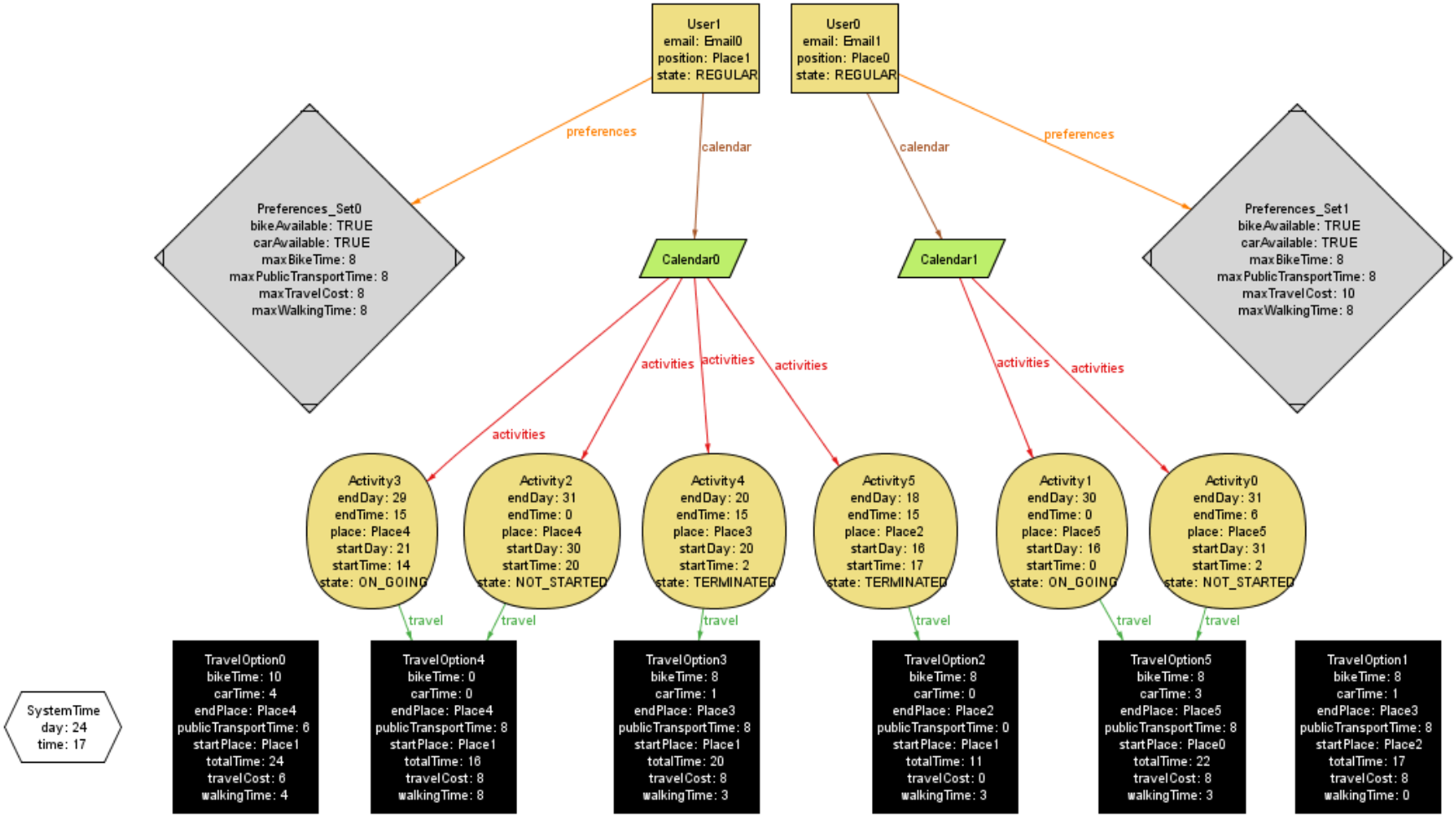
```
/*The travel option chosen for each activity must be the best one.
Among all the options that satisfy users' preferences, the best option is the one that takes less time*/
fact bestTravelOption{
      all u:User|all a:u.calendar.activities| not( some t:TravelOption |
         travelSatisfyPreferences[u,a,t] && t.totalTime<a.travel.totalTime)
}

/*Users' email addresses are unique */
fact uniqueEmails{
   all disj u1,u2:User | u1.email!=u2.email
}

/*There are no emails in the system that are not associated to a user*/
fact allEmailsAssociated{
   all e:Email| some u:User| e=u.email
}

/*There are no flexible breaks not associated to any calendar*/
fact allBreaksAssociated{
   all b:FlexibleBreak| some c:Calendar| b in c.breaks
}

/*two calendars don't share breaks*/
fact notSharedBreaks{
   all disj c1,c2:Calendar| c1.breaks & c2.breaks=none
}

/*There are no preferences without owner*/
fact allPreferencesSetsAssociated{
   all p:Preferences_Set | some u:User | p=u.preferences
}

/*There are no places not associated to an activity or a user. This is not really useful for
the representation but we need this for a matter of clarity, in order to avoid having useless places
in the diagrams*/
fact allPlacesAssociated{
   User.position + Activity.place=Place
}

/*definition of the state of an activity*/
fact activityState{
   all a:Activity|((a.state=NOT_STARTED <=>(SystemTime.day<a.startDay ||
               SystemTime.day=a.startDay && SystemTime.time<a.startTime))
      &&
   (a.state=TERMINATED <=>( SystemTime.day>a.endDay || SystemTime.day=a.endDay &&
            SystemTime.time>=a.endTime))
      &&
   (a.state=ON_GOING <=> (a.state!=TERMINATED && a.state!=NOT_STARTED)))
}

/* users can't have activities in their calendar if their email has not been confirmed*/
fact emptyCalendarIfNotConfirmed{
   all u:User | #u.calendar.activities=0 <=> u.state=EMAIL_NOT_CONFIRMED
}
```

```
/*The relationshp between users and calendars is one-to-one */
fact notSharedCalendars{
   all disj u1,u2: User | u1.calendar!=u2.calendar
}

/*different calendars don't contain the same activities (speaking about objects)*/
fact notSharedActivities{
   all disj c1,c2: Calendar | no (c1.activities & c2.activities)
}


/*all the activities are associated to a calendar*/
fact noActivitiesNotAssociated{
   all a1: Activity| some c1:Calendar| a1 in c1.activities
}

/*all the calendars are associated to a user*/

fact noCalendarsNotAssociated{
   all c1: Calendar| some u1:User| c1 in u1.calendar
}



/*in a certain calendar there are no overlapping activities, since a user can't be physically in two
different activities at the same time. This can happen when, comparing all the possible couples of
activities, the end day of one comes before the start day of the other one.
If the two activities are scheduled on the same day, the end time of one must come before the start
time of the other one*/
fact noOverlappingActivitiesOnAUser{
   all c:Calendar| all disj a1, a2:c.activities | (a1.endDay=a2.startDay && (a1.endTime<=a2.startTime ||
            a2.endTime<=a1.startTime)) ||  a1.endDay<a2.startDay || a2.endDay<a1.startDay
}


/*flexible breaks that are associated to the same calendar don't overlap*/
fact noOverlappingBreaksOnUser{
   all u:User, disj b1,b2: u.calendar.breaks |(b1.day!=b2.day ||
      b1.day=b2.day &&(
         sub[b2.endTime,b1.startTime]>=add[b1.duration,b2.duration]
            ||
         sub[b1.endTime,b2.startTime]>=add[b1.duration,b2.duration]))
}

/*flexible breaks must be coherent with activities. This means they don't have to overlap completely*/
fact flexibleBreaksDontOverlapWithActivities{
   (  all u: User, a1 : u.calendar.activities, b: u.calendar.breaks| not (
         a1.startDay<b.day && a1.endDay>b.day ||
         a1.startDay=b.day && a1.endDay>b.day && sub[a1.startTime,b.startTime]<b.duration ||
         a1.startDay<b.day && a1.endDay=b.day && sub[b.endTime,a1.endTime]<b.duration ||
         a1.startDay=b.day && a1.endDay=b.day &&
            sub[a1.startTime,b.startTime]<b.duration &&
            sub[b.endTime,a1.endTime]<b.duration)
   )
```

```
    &&
  (all u: User, disj a1,a2 : u.calendar.activities, b: u.calendar.breaks|(
    (a1.endDay<=a2.startDay && (a1.endDay=a2.startDay implies a1.endTime<=a2.startTime))
      implies
    (((a1.endDay=a2.startDay && a1.endDay=b.day &&  a1.startDay=a1.endDay &&
      a2.endDay=a2.startDay)
          implies
      (sub[a2.startTime,a1.endTime]>=b.duration && b.startTime<=a1.endTime &&
      b.endTime>=a2.startTime || sub[a1.startTime, b.startTime]>=b.duration ||
      sub[b.endTime,a2.endTime]>=b.duration))

        &&

      ((a1.endDay=a2.startDay && a1.endDay=b.day &&  a1.startDay=a1.endDay &&
      a2.endDay!=a2.startDay)
          implies
      (sub[a2.startTime,a1.endTime]>=b.duration && b.startTime<=a1.endTime &&
      b.endTime>=a2.startTime || sub[a1.startTime, b.startTime]>=b.duration))

        &&

      ((a1.endDay=a2.startDay && a1.endDay=b.day &&  a1.startDay!=a1.endDay &&
      a2.endDay=a2.startDay)
          implies
      (sub[a2.startTime,a1.endTime]>=b.duration && b.startTime<=a1.endTime &&
      b.endTime>=a2.startTime || sub[b.endTime,a2.endTime]>=b.duration))

        &&

      ((a1.endDay=a2.startDay && a1.endDay=b.day &&  a1.startDay!=a1.endDay &&
      a2.endDay!=a2.startDay)
          implies
      (sub[a2.startTime,a1.endTime]>=b.duration && b.startTime<=a1.endTime &&
      b.endTime>=a2.startTime)))))
}


pred show{
  #User=1
  #Activity=2
  #Place=2
  #TravelOption=4
  #FlexibleBreak=2
}
run show for 6 but 7 Int

/*Travel option satisfy preferences for a certain activity and a certain user*/
pred travelSatisfyPreferences[u: User, a:Activity, t:TravelOption]{
      t.startPlace=u.position && t.endPlace=a.place &&
      t.walkingTime<=u.preferences.maxWalkingTime &&
      t.bikeTime<=u.preferences.maxBikeTime &&
      t.carTime<=u.preferences.maxCarTime &&
      t.publicTransportTime<=u.preferences.maxPublicTransportTime &&
      t.travelCost<=u.preferences.maxTravelCost &&
      (u.preferences.carAvailable=FALSE implies t.carTime=0) &&
      (u.preferences.bikeAvailable=FALSE implies t.bikeTime=0)
}
```

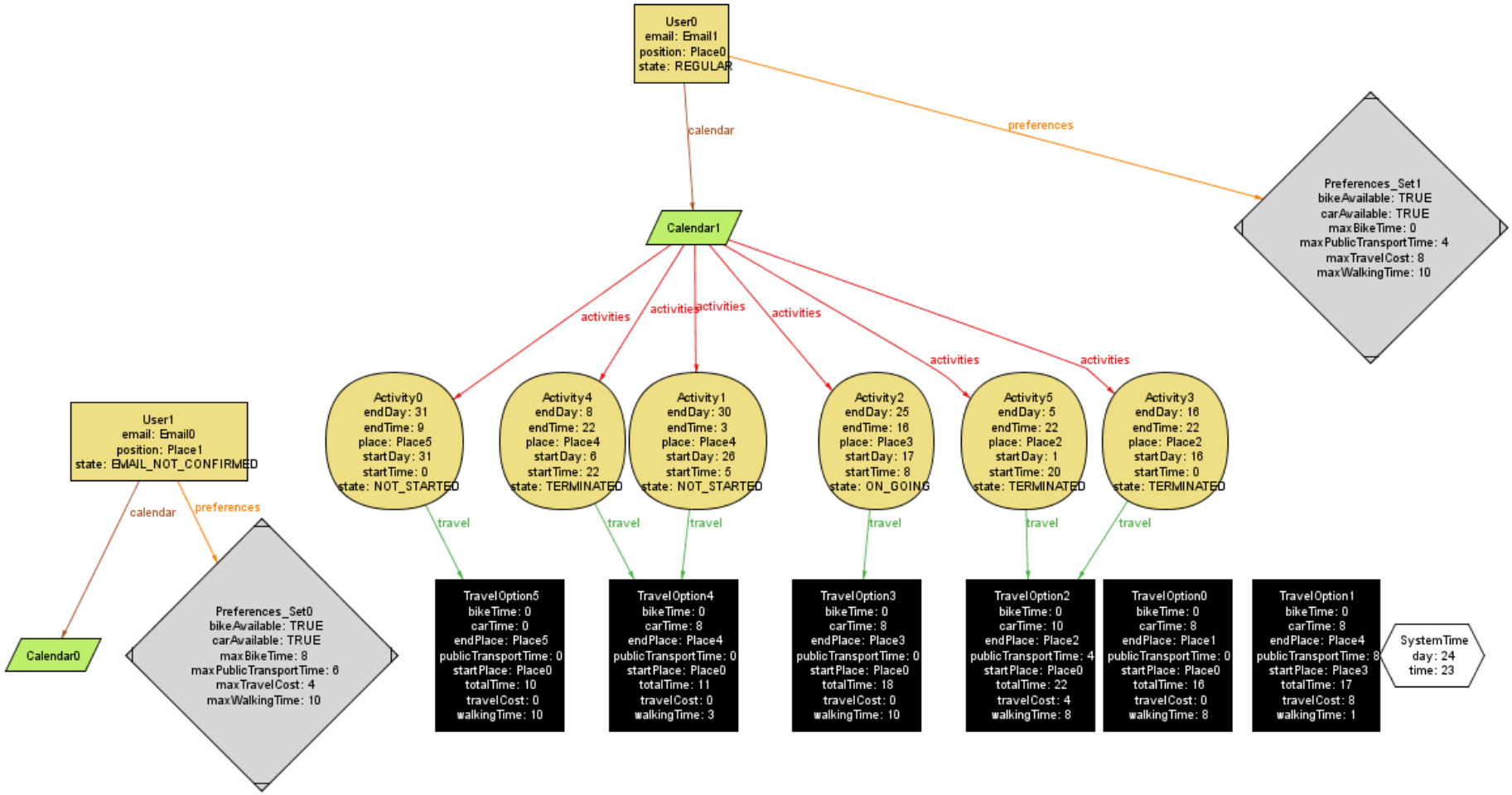
5.2 Possible Worlds

[PW1]



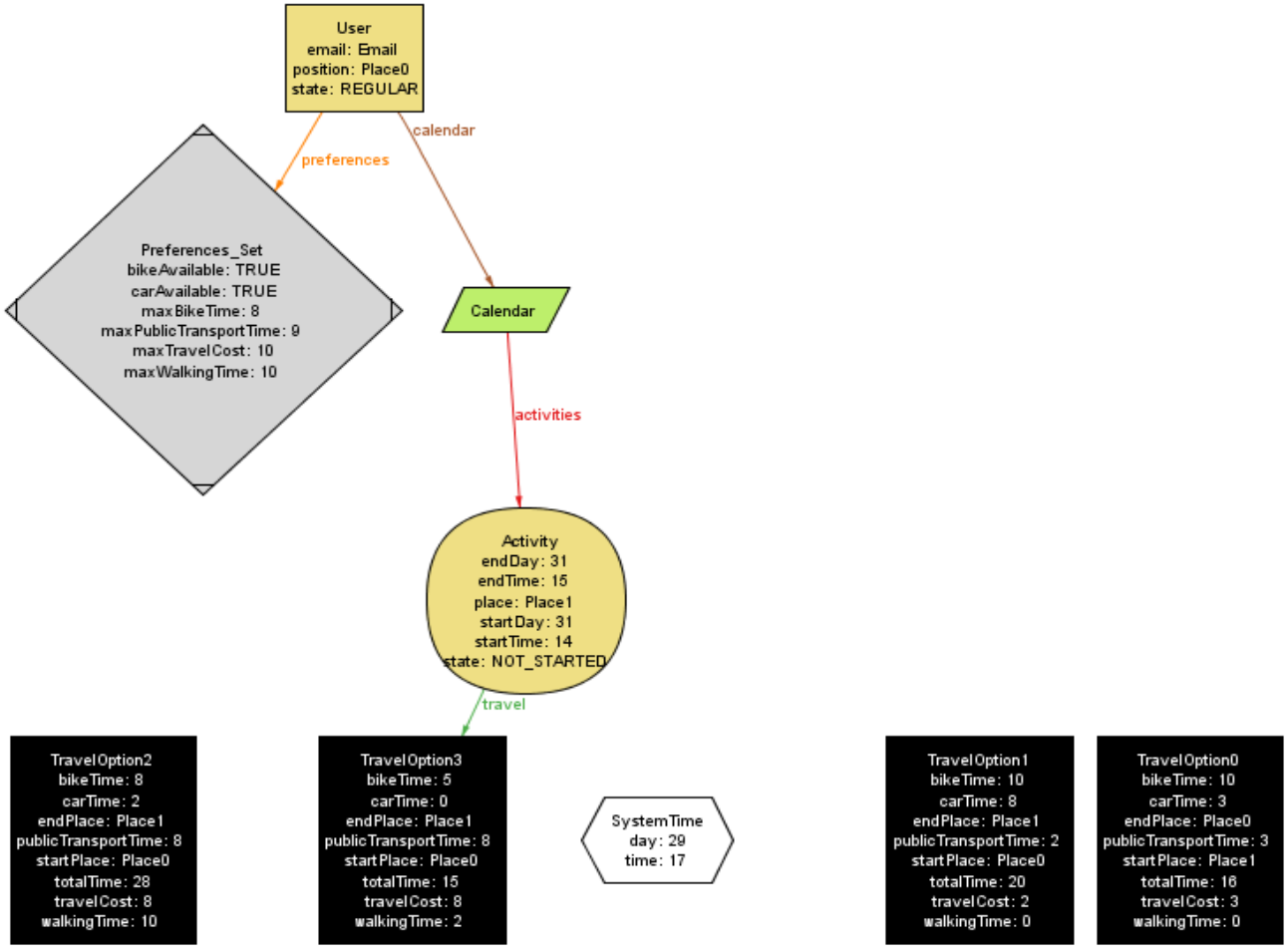
It's easily noticeable here that users whose subscription state is REGULAR can have activities in their calendars. Each activity is associated to a travel option that is compliant with the preferences. Activities' state is correctly determined.

[PW2]



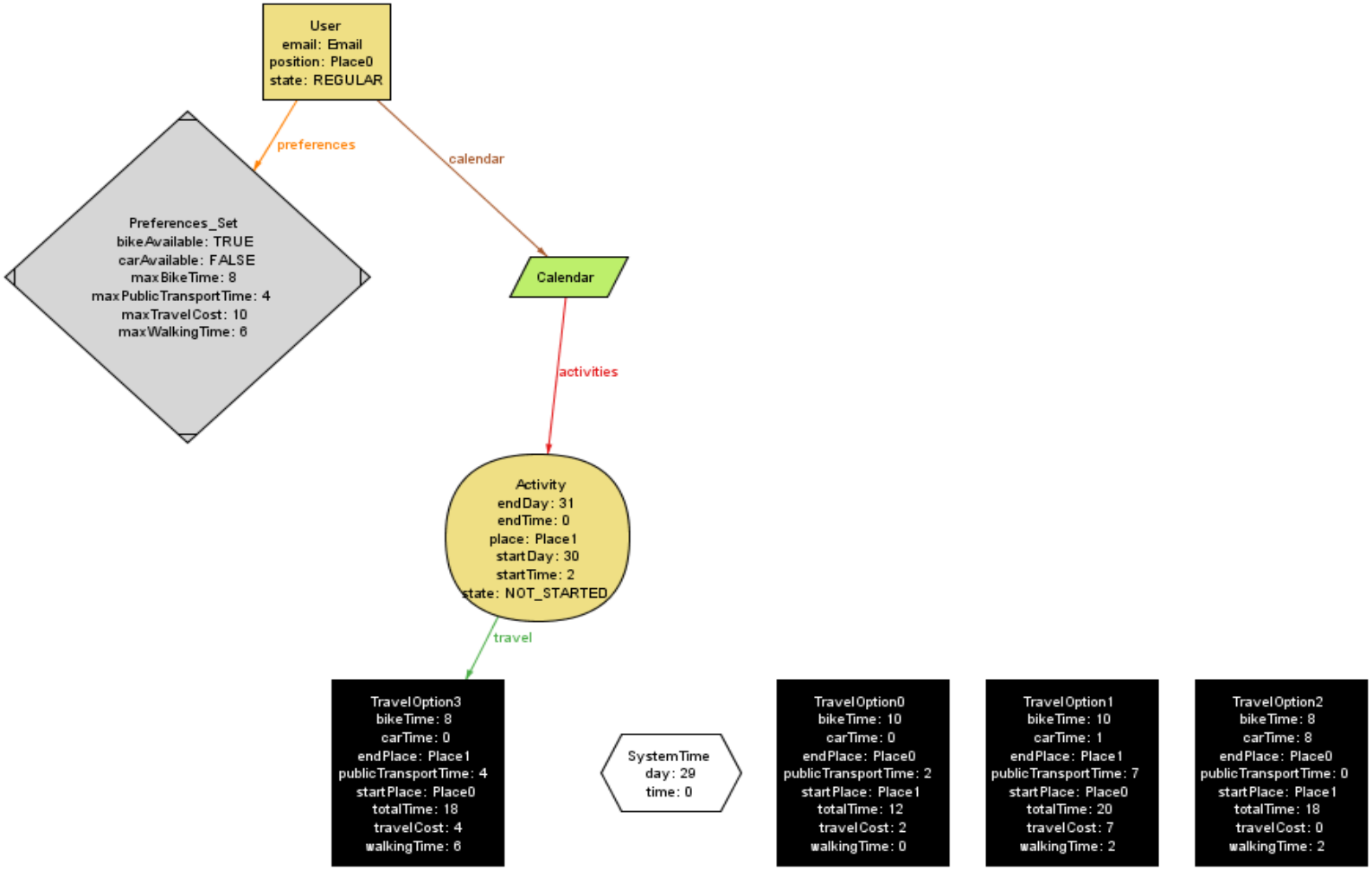
Emails are unique and users whose registration state is EMAIL_NOT_CONFIRMED cannot have activities in their calendar.

[PW3]



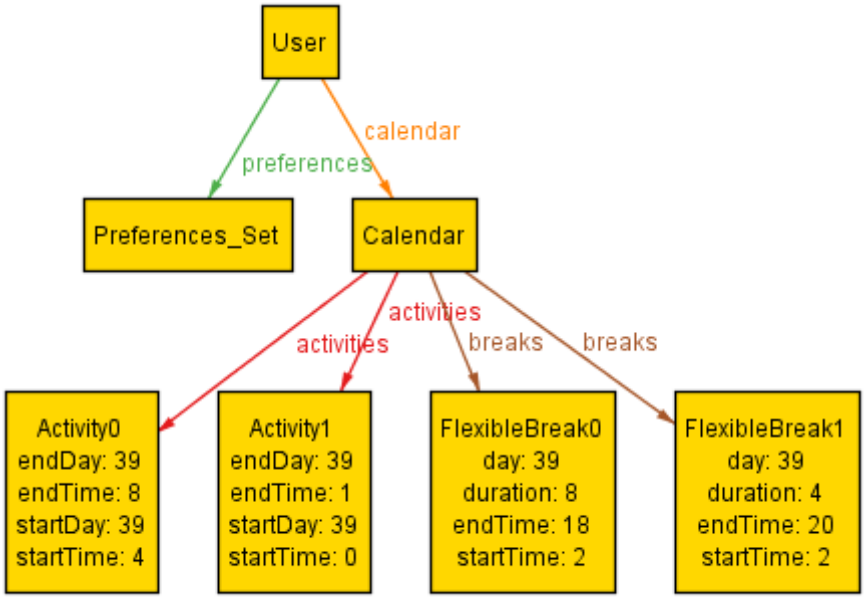
Though there are more travel options that match the preferences (TravelOption2 and TravelOption3), the chosen one is the one with the least totalTime.

[PW4]



If a user cannot use a specific transportation (in this case carAvailable=FALSE), he's given a solution that doesn't involve it. Plus, if the user sets more preferences about the same transportation (e.g. carAvailable and maxCarTime) the most restrictive one is taken into consideration (in this case carAvailable).

[PW5]



As it's possible to see in this possible world, where not relevant data have been removed, users can also schedule flexible breaks in their calendar and, inside the window [startTime, endTime], a smaller amount of time [duration] will be reserved. In this example, there are 2 fixed activities on the same day (day 39): one goes from time 0 to 1 and one goes from time 4 to 8. A possible schedule for the flexible breaks is that for FlexibleBreak0 the time reserved goes from time 8 to 16 and for FlexibleBreak1 the time reserved goes from 16 to 20, which is perfectly coherent with the requirements.

6. Effort Spent

This section will provide detailed information about the number of hours spent on this document.

Matteo Biasielli, matr. 893590

Section(s)	**Number of hours**
5-oct-17 Introduction	3
7-oct-17 Overall Description	1
8-oct-17 Overall Description	1
9-oct-17 Group work	3
10-oct-17 Requirements	2
10-oct-17 Statechart	1
11-oct-17 BPMN	2
12-oct-17 BPMN	1,5
14-oct-17 Group work	3
14-oct-17 Alloy	2,5
15-oct-17 Alloy	2,5
16-oct-17 Alloy	1
17-oct-17 Alloy	2
18-oct-17 Review	2
20-oct-17 Included alloy + possible worlds	1,5
21-oct-17 General revision	1
23-oct-17 Merged works	1
24-oct-17 Alloy	2
25-oct-17 Alloy	1
28-oct-17 Group Work	3
TOTAL:	37

Mattia Di Fatta, matr. 893608

6-oct-17 Introduction	2
7-oct-17 Introduction	2.5
8-oct-17 Specific requirements	1
9-oct-17 Group work	3
10-oct-17 Interfaces and start UC	2.5
11-oct-17 Add new use cases	2.5
12-oct-17 Complete Use Cases tables	2
13-oct-17 Start Class diagram	1
14-oct-17 Group work	3
15-oct-17 Interface revision	1,5
17-oct-17 Software System Attributes	1,5
22-oct-17 Group revision	3
25-oct-17 Minor changes in several sections	1.5
28-oct-17 Group Work	3
TOTAL:	30

Emilio Capo, matr. 899842

6-oct-17 Introduction	3
8-oct-17 Overall Description	1
9-oct-17 Group work	3
11-oct-17 UI Hand drawn design	1,5
12-oct-17 UI Hand drawn design	5
13-oct-17 UI Digital design	4
14-oct-17 Group work	3
14-oct-17 UI Digital design	4
15-oct-17 UI Digital design	3,5
16-oct-17 UI Digital design	2
17-oct-17 UI Digital design +BPMN	1
18-oct-17 BPMN	1
21-oct-17 BPMN + UI Description	1,5
22-oct-17 BPMN Description + group revision	3
28-oct-17 Group Work	3
TOTAL:	39