

Politecnico di Milano

A.Y. 2017/2018

Software Engineering 2: ***Travlendar+***

**Requirements Analysis and Specification Document**

Matteo Biasielli - Emilio Capo - Mattia Di Fatta

v. 1.0

**Table of Contents**

**1. Introduction**

1.1. Document purpose..………………………………………………...………………….3

1.2. Description of the problem………………………………………………………….3

1.3. Actors………………………………………………………………………………………...3

1.4. Goals…………………………………………………………………………………………..4

1.5. Definitions, Acronyms, Abbreviations………………………………………….4

1.5.1 Definitions……………………………………………………………………….4

1.5.2 Acronyms………………………………………………………………………...4

1.5.3 Abbreviations…………………………………………………………………..4

1.6. Reference Documents…………………………………………………………………5

1.7. Document Structure……………………………………………………………………5

1.8. Revision History…………………………………………………………………………5

**2. Overall Description**

2.1. Product Perspective…...……………………………………………………………….6

2.2. User Characteristics……………………………………………………………………6

2.3. Assumptions, Dependencies and Constraints……………………………….6

2.3.1 Assumptions……………………………………………………………………6

2.3.2 Constraints……………………………………………………………………...7

**3. Specific Requirements**

3.1. Functional Requirements……………………………………………………………8

3.2. Non-Functional Requirements…………………………………………………….8

3.3. Requirements and Assumptions Mapping……………………………………9

3.4. Reasons that justify some requirements……………..……………………….9

3.5. External Interface Requirements………………………………………………..10

3.5.1 Software Interfaces…………………………………………………………10

3.5.2 Communication Interfaces………………………………………………10

3.6. UML diagrams and charts…………………………………………………………..11

3.6.1 Use Cases Diagram and tables…………………………………………11

3.6.2 Class Diagram…………………………………………………………………12

3.6.3 Statechart Diagrams………………………………………………………..18

3.6.4 BPMN Diagrams……………………………………………………………...20

3.7. Graphical User Interface Examples

**1. Introduction**

* 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. **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 should 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. **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. **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 an new activity.
* [G5] Make sure users are on time at their scheduled appointments.
  1. **Definitions, Acronyms, Abbreviations**
     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 both the desktop version and mobile version of the Travlendar+ system.
* **Scheduling**: action performed by a user that is adding a new activity to his personal calendar.
* **Activity**: whatever the final user wants to schedule, adding it to the calendar and providing the app with its information.
  + 1. **Acronyms**
* **RASD:** Requirements Analysis and Specification Document
* **UI:** User Interface
* **API:** [Application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface)
* **UXD**: User Experience Diagram
* **UML**: Unified Modeling Language
* **GPS**: Global Positioning System
  + 1. **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
  1. **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. **Document Structure**
* **Introduction:** This is the very first part of the document.

In this section it’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.
* **Effort Spent:** Information about the number of hours each group member has spent working on the RASD document.
  1. **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. 1.0 [09 Oct 2017]:** general group revision of the first two sections.
* **V. 1.1 [14 Oct 2017]:** Added specific requirements.

**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.

* 1. **Assumptions, Dependencies and Constraints**
     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.
* **Assumption local initial app?**
  + 1. **Constraints**
* **[C1]** Confidential data inserted by the users must be stored a secure way, according to the 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**

* 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 special preference “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).
  1. **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 at least on two of the following OS: Android, iOS, 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.
  1. **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 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].

* 1. **Requirements and Assumptions Mapping**

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

* [NFR1] – Many application 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 opition.
  1. **External Interfaces Requirements**
     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](https://developers.google.com/maps/documentation/geolocation/intro)) APIs will be used. Additionally, in order to store users’ login credentials and calendar’s data, 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.

Furthermore, the system needs to interface with some third part apps and services:

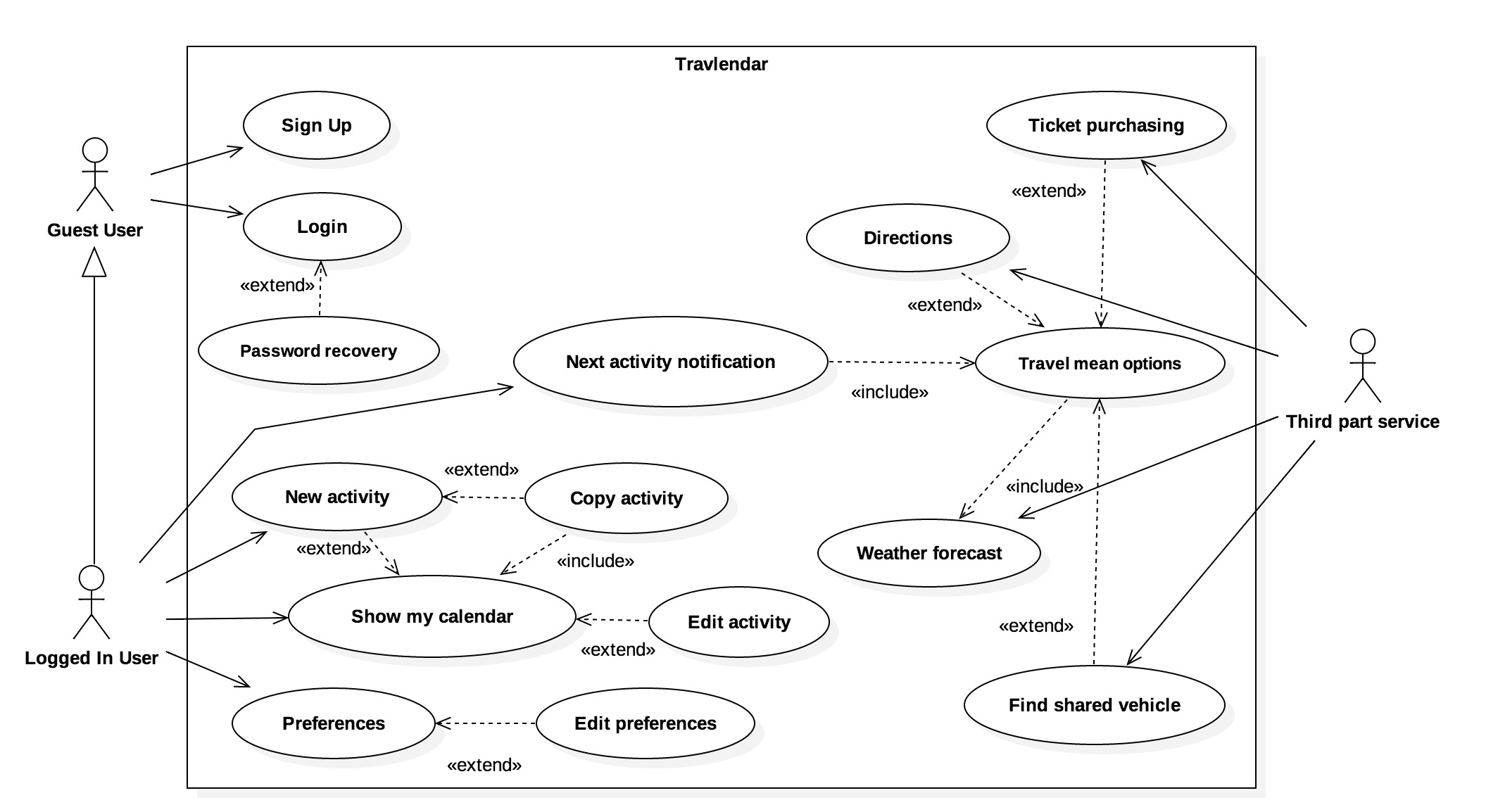
* a weather forecast service in order to provide more advanced and accurate suggestions and instructions. Yahoo! Weather APIs will be used (<https://developer.yahoo.com/weather/> ).
* Third part bike sharing apps and websites such as Ofo and Mobike, for the Milan Metropolitan Area, to be integrated in the application.
* Car sharing service Car2Go and EniEnjoi 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+.

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: Safari, Google Chrome, Internet Edge, Safari).

* + 1. **Communication Interfaces**

The entire system will make use of TCP-IP protocol for Internet communication with the web server, the DBMS and users.

* 1. **UML diagrams and charts**
     1. **Use Cases Diagrams and tables**



The use case diagram above presents all possible 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, to a “Premium” mode/account 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 user interface samples) * 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* | Moreover, if either the password provided is not correct or the username does not exist, the system notifies the mistake. |

|  |  |
| --- | --- |
| *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 homepage is displayed (for further information see user interface samples) * 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* | If the user decides to interrupt the routine, the application homepage is showed again.  Moreover, if any mandatory information to complete the registration is incorrect or missing, an error screen is displayed. |

**[UC2]** Sign Up

**[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 user interface samples) * 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 user rolls back to the application homepage, interrupting the password recovery routine.  -The link attached to the email expires and the process is interrupted. |

**[UC4]** New 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 user interface samples) * 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. |

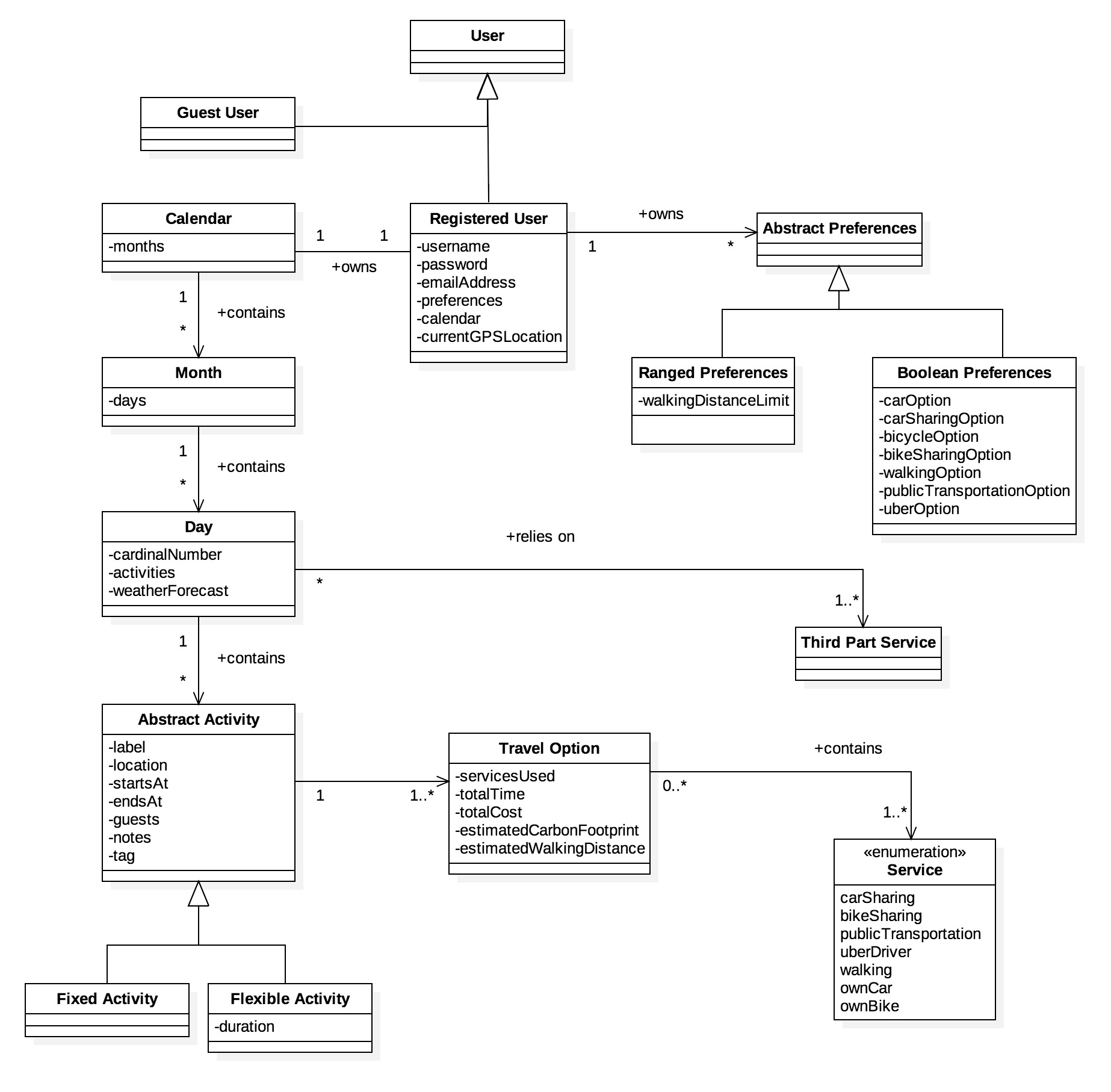
**[UC5]** Edit activity

|  |  |
| --- | --- |
| *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 user interface samples) * 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) or the user himself interrupts the edit procedure. |

**[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 user interface samples) * 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 and activity scheduling (insert/delete lunch break) * If changes are performed, the system saves them in its server |
| *Exit condition* | The system notifies the user the end of the procedure and that changes have been saved. |
| *Exceptions* | The user exits the edit procedure without saving the changes. |

* + 1. **Class Diagram**

The class diagram above contains all 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.

* + 1. **Statechart 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.

* + 1. **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.

* 1. **Graphical User Interface Examples**

**5. Effort Spent**

This section will provide detailed information about the nomver 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 |

**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 Use case and class diagram | 1.5 |

**Emilio,** matr. 899842

|  |  |
| --- | --- |
| 6-oct-17 Introduction | 3 |
| 8-oct-17 Overall Description | 1 |
| 9-oct-17 Group work | 3 |
| 11-oct-17 UI | 1,5 |
| 12-oct-17 UI | 5 |
| 13-oct-17 UI | 4 |
| 14-oct-17 Group work | 3 |
| 14-oct-17 UI | 1,5 |