POLITECNICO
MILANO 1863

Software Engineering II

Travlendar+

Requirements Analysis
and
Specifications Document

Authors:

Edoardo D'Amico
Gabbolini Giovanni
Parroni Federico

1st October 2017

Indice

Chapter 1

Introduction

1.1 Purpose

Our team will develop *Travlendar+*, a calendar-based application that aims to provide a schedule of user appointments, giving a plan to organize his daily life. The main goals the app must fulfill are:

G1 The system should offer the possibility to create a new account;

G2 The system should be able to handle a login phase;

G3 The system should give to the signed user the possibility to recover his password;

G4 The system should allow the user to insert an appointment according to his necessities and his preferences (1.3.1);

G5 The system S.P.W. to modify an inserted appointment;

G6 The system S.P.W. to create a valid schedule (1.3.6) of the user appointments when requested and display the scheduling result (1.3.9);

G7 The system should let the user create valid multiple schedules and decide which one is chosen for the current day;

G8 The system S.B.A to book the travel means involved in the current schedule under user approval;

G9 The system S.B.A. to display in real time user position and the directions to be followed in order to arrive to the next appointment on a dinamically updated map;

G10 The system S.B.A. to notify the user when a shared travel mean is available and it would optmize the current schedule;

1.2 Scope

Here we provide a brief description of the aspects of the reality of interest which the application is going to interact with.

User can receive an appointment on a certain date, time and location (over a region), that can be reached using different available travel means. The appointment can be held either at a specific time or in a time interval and lasts for a certain amount of time. An appointment can be recurrent, in other words, it repeats regularly over time (e.g., lunch, training, etc.). User can travel with someone else and can pick up or leave off these people during the day.

User can have his own travel means and a pass for public transportation. The travel means considered in this scenario can be grouped in three categories: public, shared or private.

- Public travel means: these include trains, buses, underground, taxis, trams. They have to be taken in their designated stops. User must have a valid ticket in order to get on a public travel means (except for taxis, that pick up the user wherever he wants upon a call and do not require any ticket);
- Shared travel means: these include cars and bikes. They are located in specific places.
- Private travel means: vehicles owned by the user. They can be cars, bikes. Also walking is considered to be a (very special) private travel mean.

Weather conditions can change during the day affecting travel means choice. At the beginning of the day, or on demand, user can request a schedule of his daily appointments, following some criteria evaluated according to their assigned priority and satisfying some constraints imposed by the user. When a new appointment is received, user creates a

new item in the application and saves it in the appointment list. User can request the application to reschedule the appointments because of unexpected changes of his plan (e.g. a cancelled appointment). User can choose whether to take or not a shared travel mean when the application notify him of its availability. A daily schedule starts from an initial location that can be set or automatically retrieved by GPS and it's supposed to end in the place of the last appointment.

1.2.1 World Phenomena

- User receives a new appointment;
- User picks up a person;
- User owns private travel means and/or passes for public transportation;
- User wakes up;
- User pass expires;
- Exists various travel means.

1.2.2 Shared Phenomena

- Shared travel mean moves;
- Shared travel mean is not available anymore;
- Wheather condition changes;
- Public travel means reach a stop-place;
- Public travel means are late;
- Public travel means are not available due to a strike day;
- User requests a schedule to the machine;
- User inserts a new appointment into the application;
- User requests to book rides;
- User moves.

1.3 Definitions, Acronyms, Synonims

1.3.1 Synonims

Are synonims:

- Appointment and meeting;
- System and Application.

1.3.2 Definitions

Definition 1.3.1. A preference is a constraint on appointment or a schedule;

Definition 1.3.2. A device is a PC, a Tablet or a Smartphone in which run the last version of his O.S.;

Definition 1.3.3. A Travel Option is a combination of travel path and travel means that allow to reach one spot from another;

Definition 1.3.4. The Travel Option Data are additional information about a travel option:

- Cost;
- Traveling time;
- Carbon emission;
- Distance (KM);
- Graphical representation of the path.

Definition 1.3.5. A Schedule is a set of time-ordered and not overlapping appointments where their starting times are fixed and they're linked to each other by a path covered with a specific transportation mean;

Definition 1.3.6. A Valid Schedule is a Schedule which:

- Is optimized according to the criteria chosen by the user;
- Ensures that the user will be on time for all his appointments;

- Respects the constraints imposed by the user;

Definition 1.3.7. A travel service account is an external account of the user which permits the booking and the payment of a specific travel mean;

Definition 1.3.8. A Relative Path is a portion of a path travelled by the same travel means;

Definition 1.3.9. A Scheduling Result is the set:

- Graphical representation of the path that will be travelled by the user
- Money spent for each relative path
- Total money spent
- Length of the path expressed in KM
- Length of relative path
- Carbon footprint emission
- Estimated travel duration of each relative path
- Total estimated travel time

Definition 1.3.10. The Current Appointment is an appointment which has *startingTime* >= *currentTime* and *date*=*currentDate*, where *currentTime* and *currentDate* are the actual time reference of the system.

1.3.3 Acronyms

Acronyms used in the text:

- GPS: Global Positioning System;
- GUI: Graphical User Interface;
- ETA: Estimated Time of Arrival;
- S.P.W.: Should Provide a Way;
- S.B.A.: Should Be Able;

- API: Application Programming Interface;
- CRUD: Create/Read/Update/Delete;
- URL: Uniform Resource Locator;
- O.S.: Operating System.

1.4 Revision history

1.5 Reference documents

1.6 Document structure

Chapter 2

Overall Description

2.1 Product Perspective

2.1.1 User Model

A user is represented within the application by his password and his e-mail. Some important informations about him are held by the following parameters:

- *travelPass*: indicates if the user has a pass for public transportation;
- *hasBike*: indicates if the user has his own car;
- *hasCar*: indicates if the user has his own bicycle;
- *externalTravelServicesAccounts* (1.3.7);

2.1.2 Appointment Model

An appointment is represented within the application by a set of parameters:

- *duration*: the time extension of the appointment;
- *date*: the day in which the appointment is held;
- *startingTime* or *timeInterval*: the first should be given if the starting hour is well-known (deterministic), otherwise a time interval in which the appointment will be held it's provided;
- *location*: identifies the coordinates of the place where the appointment will be held;

- *recurrent*: specifies if the appointment will be repeated over a fixed period of time;
- *peopleTravelling*: represents a variation of people occourring when the user picks up or leaves off someone.

The life cycle of an appointment can be represented by the following statechart:

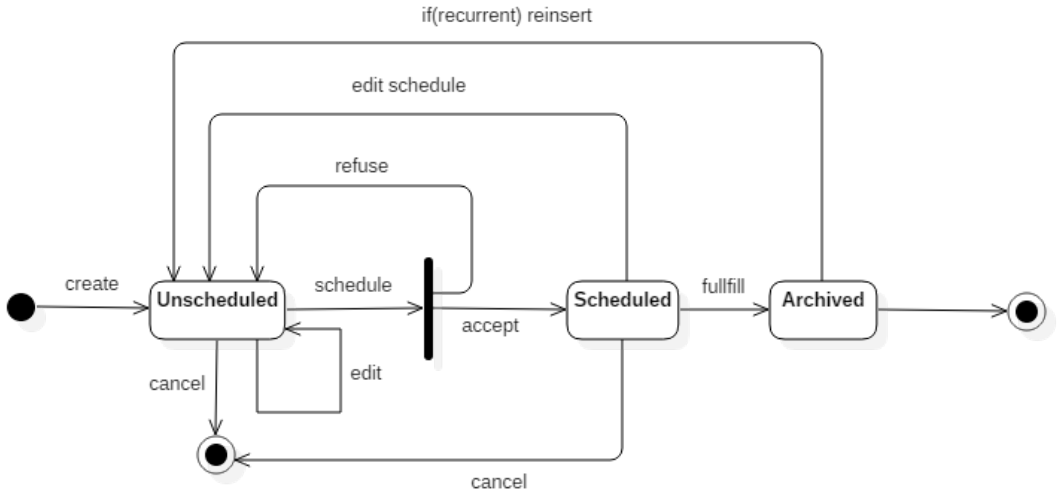


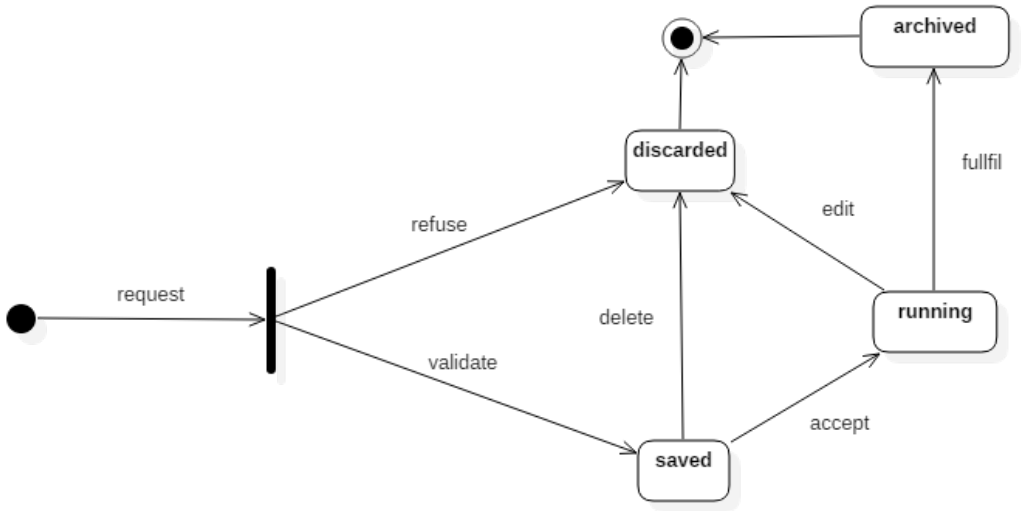
Figura 2.1: Appointment statechart

A newly-created appointment is **Unscheduled**. It could remain **Unscheduled** either when edited or there isn't a possible arrangement when a schedule is performed. Otherwise it becomes **Scheduled** if there's a feasible way to arrange it. When a scheduled appointment is edited all the appointments in that schedule return to be **Unscheduled**, because they can possibly cause a different schedule. When a scheduled appointment is fullfilled it becomes **Archived** and stored in the schedule history. If this last one is a recurrent appointment it must be reinserted in the list of unscheduled appointments so it will become **Unscheduled** again. The user can cancel an appointment in every moment.

2.1.3 Schedule Model

A schedule is a set of Appointments of a given day, ordered by the scheduler following the criteria described below. A schedule is characterized by the following variables:

- *date*;
- *startingPosition*: is the starting location of the user (e.g. user's home);
- *startingNumberOfPeople*: the number of people that must reach the first appointment;
- *wakeUpTime*: it is the starting time from which the schedule should start arranging appointments.



When a schedule is requested by the user it can be either validated or refused by the scheduler according to the definition of valid schedule (1.3.6). In the first case the schedule is **saved** in the second one the schedule is **discarded** and a warning to the user is sent. In order to start a schedule the user must accept one of them from the saved ones, after that the schedule is **running**.If the user edits one of the appointments belonging to the running schedule, this one is not more valid and become **discarded**. When a schedile is fullfilled by the user this last is **archived** by the system.

The optimization criteria

Criteria for the schedule optimization that can be chosen by the user. These are the following:

- *Minimize carbon footprint*: the scheduler will try to minimize the emission of polluting gases;

- *Minimize money spent*: the scheduler will try to avoid expensive means and to exploit the cheap ones so that the amount of money used will be minimun;
- *Minimize travelling time*: the scheduler will compute the quickest possible path reaching all the appointments locations.

2.1.4 Constraints

Constraints are impositions on some parameters managed by the system during the process of scheduling the appointments. We can distinguish between constraints on schedule and contraints on the single appointment. These can be selected by the user when he inserts an appointment or when he requests a schedule, otherwise the constraints are initialized to default values.

Constraints on schedule

- *Maximum travelling distance with a specific travel mean*: the user can set a maximum amount of km to travel with a specific travel mean;
- *Travel means time slots*: user can specify a time interval in which a travel mean can be used;
- *User can deactivate a particular travel mean*;
- *User can select which travel mean he uses under certain weather condition.*

The system checks the availability of shared travel means 15 minutes before the beginning of an appointment and notifies the user only if taking that mean will provide a better solution according to the optimization criteria selected.

Constraints on appointment

- *User can deactivate a particular travel mean.*

2.1.5 Class Diagram

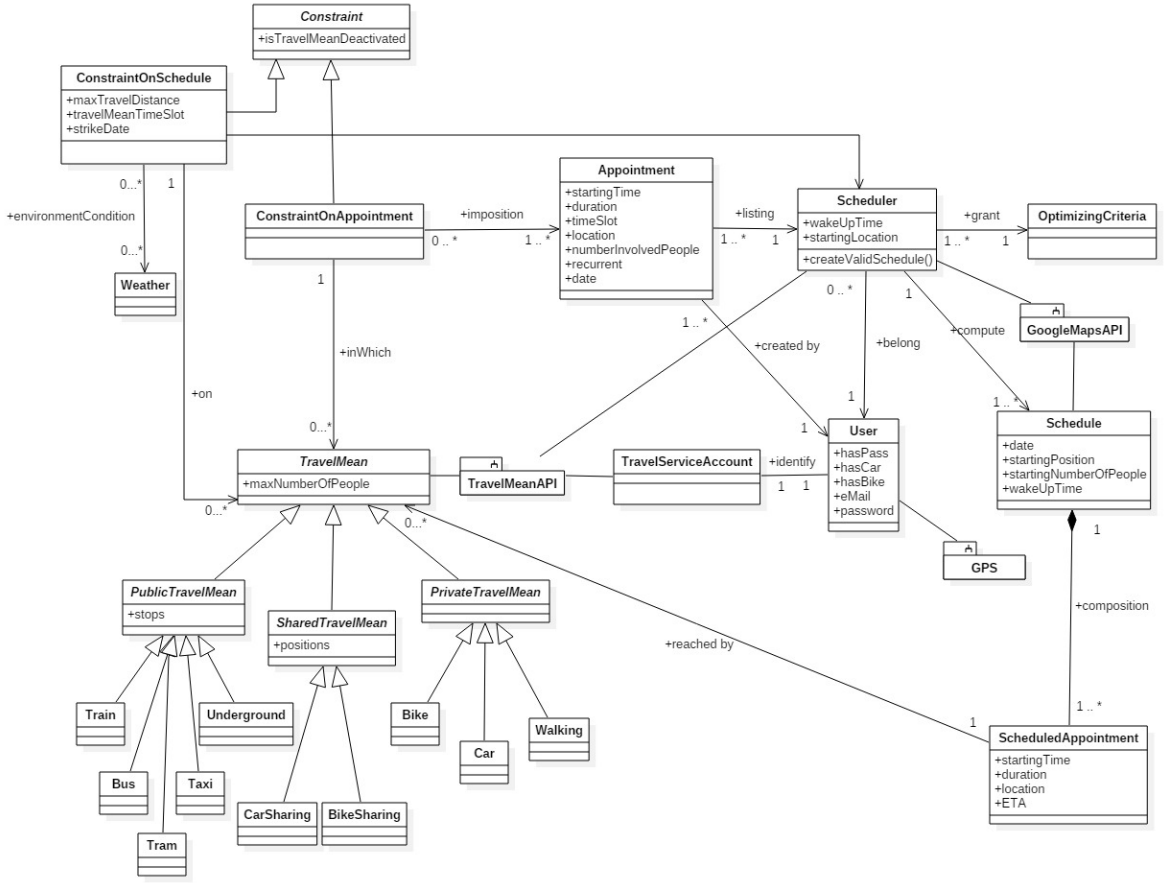


Figura 2.2: System Class Diagram

2.2 Product Functions

The following requirements are derived in order to fullfill the specified goals.

Requirements for **G2**:

R1 The system S.B.A. to retrieve information from the user about his appointments;

R2 The system S.B.A to store an appointment in his memory;

Requirements for **G??**:

R3 The system should let the user change the parameters and the constraints of an inserted appointment;

R4 The system S.B.A to rewrite the appointment in his memory with his new parameters;

Requirements for **G4**:

R5 Allow the user to set the constraints of the schedule (**riferimento ai constraint dello schedule**), or to accept the default values;

R6 Allow the user to set the optimization criteria (2.1.3) for the schedule;

R7 Allow the user to set the variables for the schedule;

R8 The system S.B.A. to gather information from external APIs about:

- travel options with related travel option data;
- weather forecast;
- strike days;

R9 The system S.B.A. to select the best travel option according to the optimization criteria taking into account:

- user constraint
- user parameters **(aggiungere riferimento a user model)**
- travel option data
- weather forecast
- information about strike day

R10 The system S.B.A to store valid schedules requested by the user

R11 The user can specify his parameters (**ref a usermodel**)

Requirements for **G5**:

R12 The system should let the user accept a schedule from the saved ones

Requirements for **G6**:

R13 The system S.B.A to handle a registration phase in which the user will provide an e-mail and a password;

R14 The system S.B.A. to verify the e-mail given by the user by sending a confirmation e-mail to his address;

Requirements for **G9**:

R15 The system S.B.A to recognize a registered user given an e-mail and a password

R16 The system S.B.A. to retrieve information from the user about his registration informations, i.e. his e-mail and password (**da verificare se serve o meno**);

Requirements for **G8**:

R17 The system should offer to the user a way to link all his travel service accounts into the Travlendar+ account;

R18 The system S.B.A to book a travel mean through external API offered by third party application in which the user is signed

Requirements for **G9**:

R19 The system S.B.A to retrive the graphical representation of a path from an external API;

R20 The stystem S.B.A to retrive the travel option from an external API **(questa che gia si è detta ma serve anche per questo goal come facciamo?)**

R21 The S.B.A to retrieve the length of a path from an external API

Requirements for **G10**:

R22 The system S.B.A to retrive the position of the user from his GPS

R23 The system S.B.A. to retrive from an external API the directions to give to the user for reach the next appointment;

Requirements for **G??**:

R24 The system should be able to recover an e-mail address from the user;

R25 The system should be able to send the password of a user to his e-mail given the e-mail.

Requirements for **G??**:

R26 The system should be able to send notifications to the user;

R27 The system should be able to retrieve information about the availablity of shared travel means from external APIs without the user request

2.2.1 User characteristics

Users can use our system when they want something that allows them to schedule their meetings according to their necessities and constraints. Necessary conditions for the users in order to use the system are:

- He must have a device connected to the internet in which the application runs

This is the only requirement that is needed. Anyway additional characteristics of the user lead to the exploitment of all the system features. In fact some of them are guaranteed only after having submitted some information to the application. In this sense, welcomed user's characteristics are:

- The ownership of some travel means
- The ownership of travel passes
- The registration to sharing services

Beside these, an obvious tacit assumption is that the user has a valid age to move where he wants with autonomy.

2.2.2 Assumptions, dependencies and constraints

- Shared Travel Means require a reservation at least 15 minutes in advantage in order to be used by the user;
- The system should be able to retrieve information about public travel means. In particular:
  - information about delays;

  - information on possible strike days.

- The system should be able to retrieve information about shared means. In particular:
  - position of the available ones;
  - prices per time unit;
- There exist external APIs that allow to:
  - retrieve all travel options and travel option data
  - signed user to book and pay for all travel services
  - retrieve information about weather forecast
  - retrieve a graphical representation of a path
  - retrieve the lenght of a path
- The device on which the application runs is connected to the internet;
- The user has a GPS active in every moment;
- Every user has at least one personal e-mail;
- Shared travel means are only cars and bikes;
- Shared travel means can be booked for the next 15 minutes (**dire due parole su questa assunzione che è abbanstanza forte**);

Chapter 3

Specific requirements

3.1 External Interface Requirements

The application shows its best potential when running in a mobile device, for instance a smartphone or a tablet. This permits to extend the features and the automatic tasks of the application, thanks to the built-in device functionalities. However, a computer client version of the application can be installed, too.

3.1.1 User interfaces

The user can interact with the application through several graphical interfaces:

1. **Registration/login interface**: allows the user to insert credentials in order to registering or logging into the system;

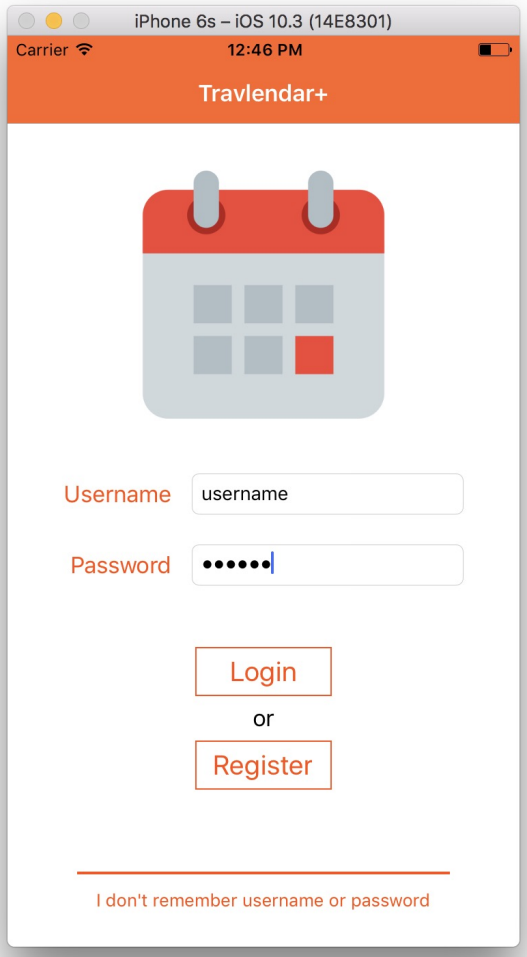


Figura 3.1: Registration/login interface

2. **User account interface**: user can specify his profile characteristics, such as his passes, car and/or bike ownership;

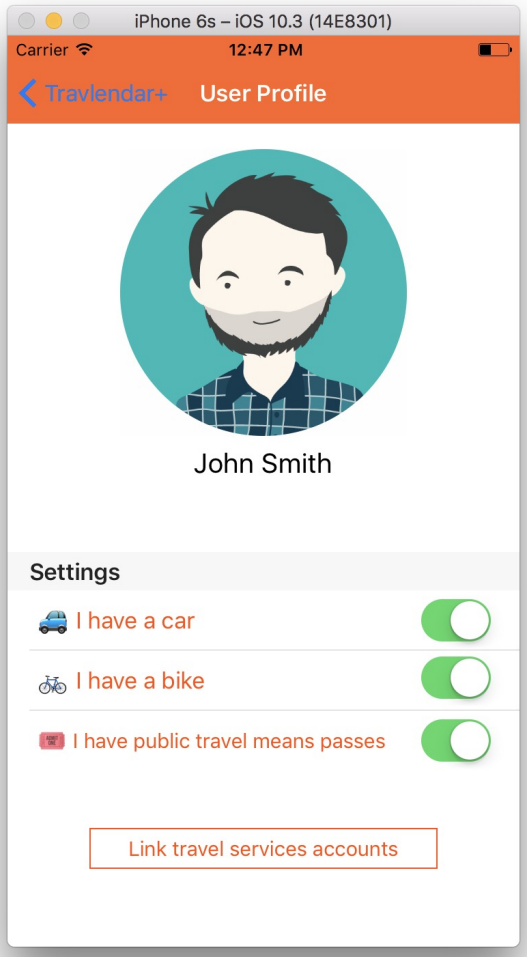


Figura 3.2: User account interface

3. **Home interface**: shows currently running schedule and displays some navigation links to other interfaces;

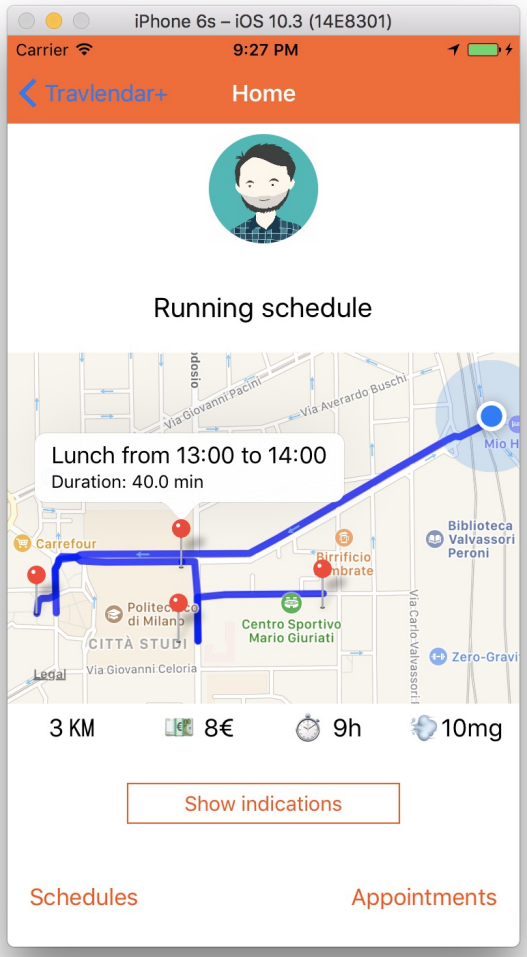


Figura 3.3: Home interface

4. **Appointment CRUD interface**: allows creating, showing and editing appointment parameters and related constraints;

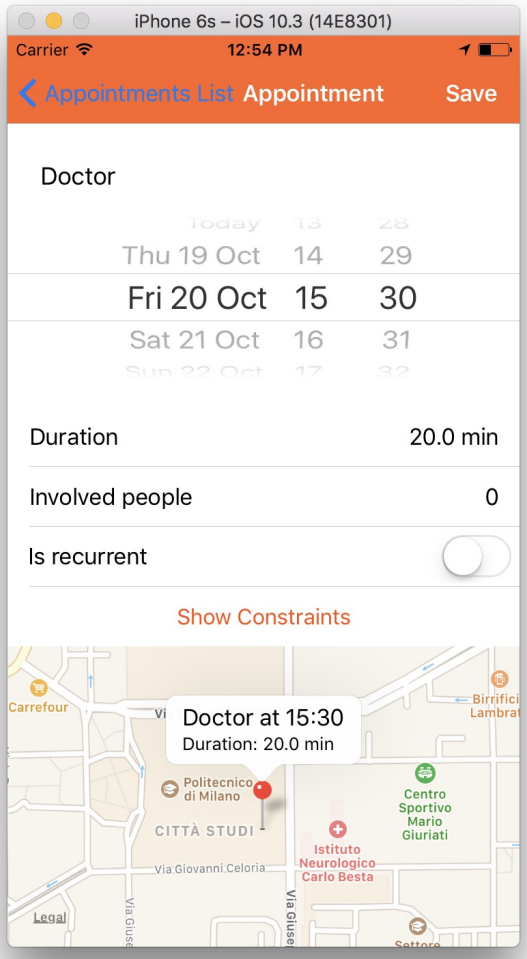


Figura 3.4: Appointment CRUD interface

5. **Appointments list interface**: provides a list of all inserted appointments, with the possibility to filter between non-scheduled/scheduled ones (includes the possibility to delete an item of the list);

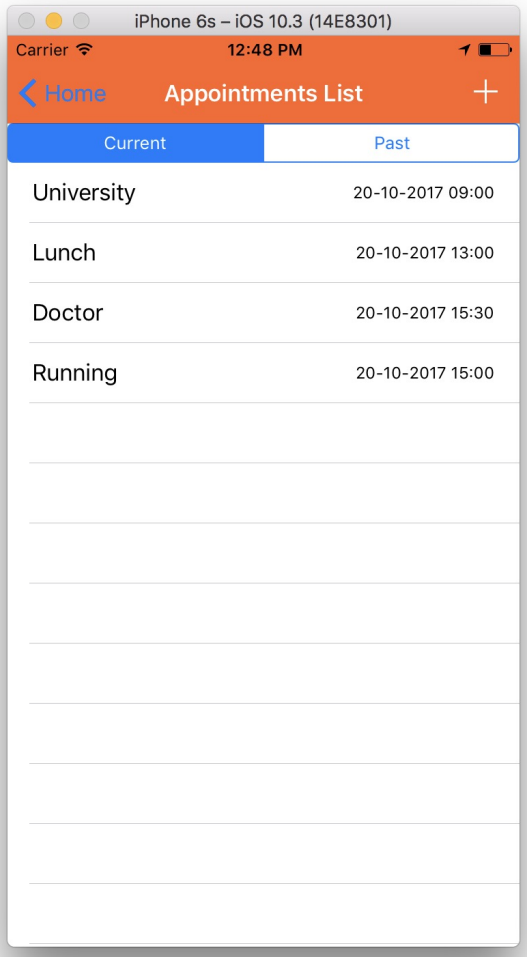


Figura 3.5: Appointments list interface

6. **Schedules list interface**: display a list of the created schedules

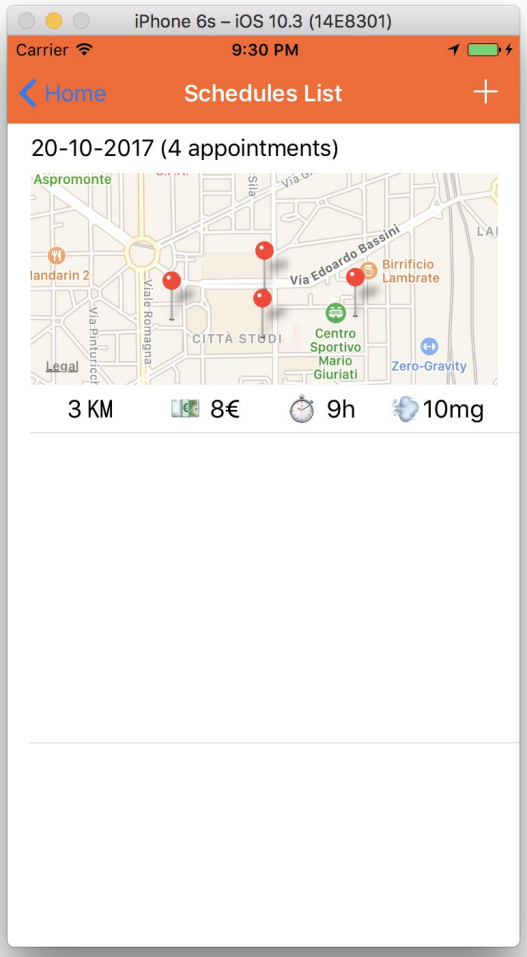


Figura 3.6: Schedules list interface

7. **Schedule interface**: user can set parameters, contraints, optimization criteria and request a schedule creation for a given date;

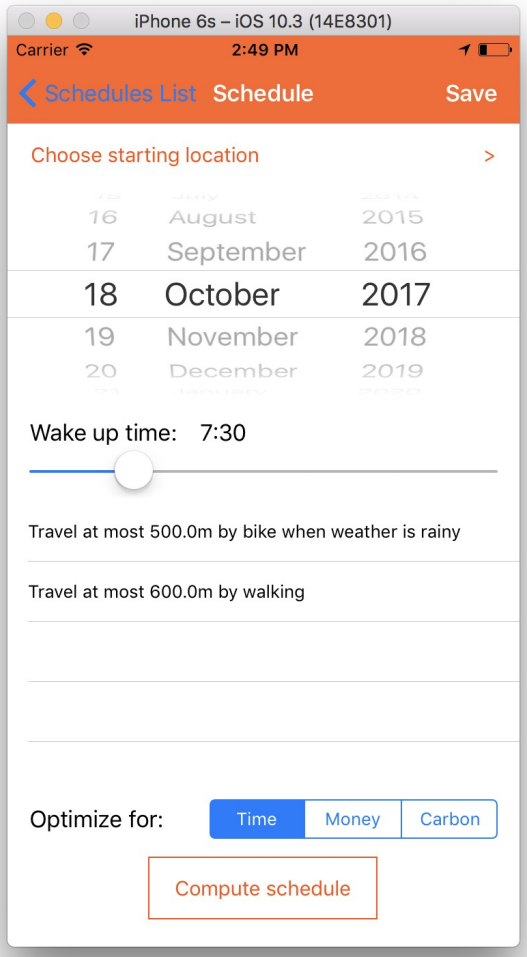
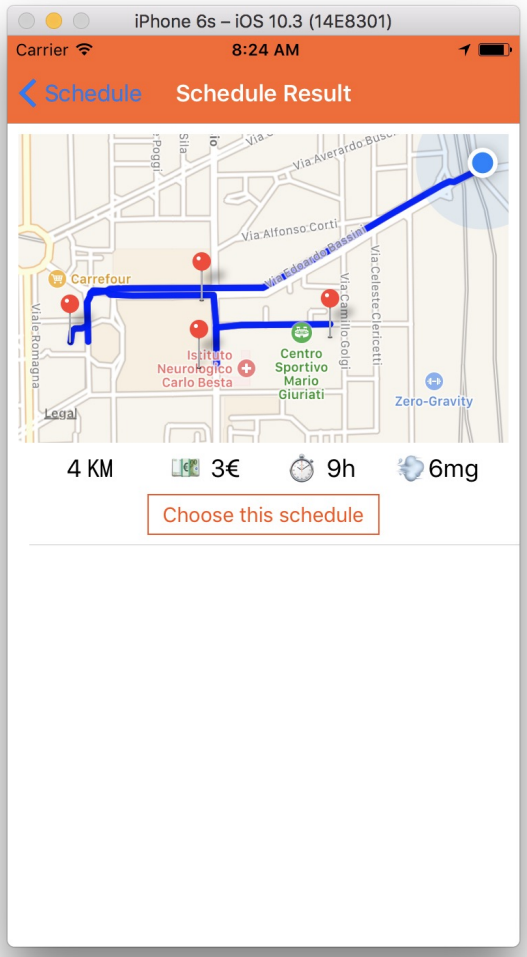


Figura 3.7: Schedule interface

8. **Schedules result interface**: shows the computation of the requested schedules for a given date and asks the user to select one, then waits for confirmation for that;



results".jpg

Figura 3.8: Schedules result interface

9. **Schedule progress interface**: permits to keep track of the completeness percentage, indicating the directions to be followed by the user in a map, in order to arrive to the next appointments;

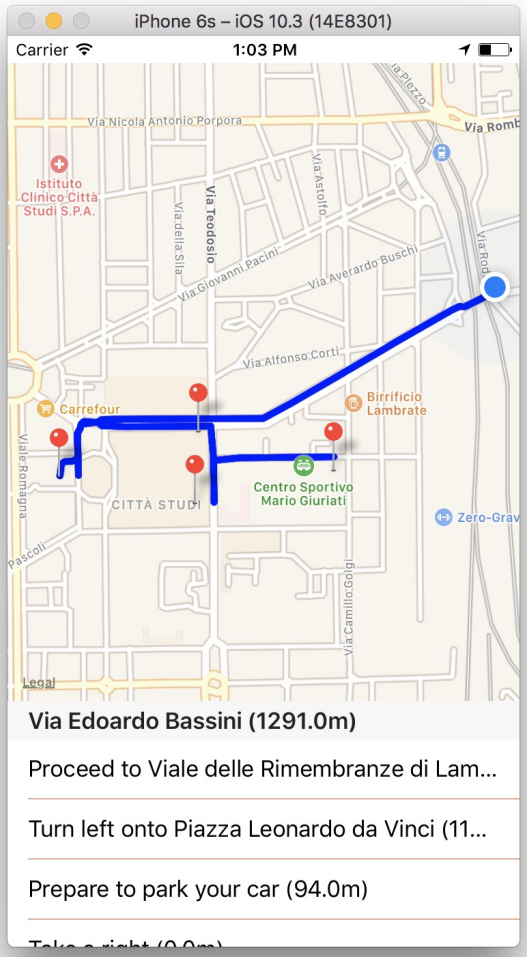


Figura 3.9: Schedule progress interface

10. **Tickets/rides reservation interface**: allows user to buy tickets for public travel means and/or reserve a ride for the shared travel means;

3.1.2 Hardware interfaces

Hardware interfaces are physical linking across which two or more separate components of a system exchange information. A hardware interface is described by the mechanical and electrical signals at the interface and the protocol for sequencing them. There are no interesting hardware interfaces in our scope.

3.1.3 Software interfaces

Software interfaces are logical linking across which two or more separate applications running on a system exchange information. The most relevant software interface in our system is API. APIs are sets of subroutine definitions, protocols and clearly defined methods of communication, allowing data exchanging and service requests. There are several kinds of these:

- **Operating System APIs**: specify interface between applications and OS, permitting to access low level routines calls (for instance, to communicate with memory or with an internal device)**(!)**
- **Remote APIs**: DBMS expose a set of standards that the API user can adopt in order to manage the database data. SQL is the standard language for storing, manipulating and retrieving data in this context;
- **Web API**: information can be exchanged through the internet by encapsulating it in HTTP request/response. Weather forecast, travel services, mapping systems offer this typology of API.

3.1.4 Communications interfaces

Communication interfaces allows two different architectures of the system to exchange information through communication channel. These non-homogeneous components of the system can communicate thanks to the following software interfaces and protocols:

- **Cellular connectivity**: mobile devices can connect to the internet thanks to LTE standard;
- **GPS**: cellular can retrieve his coordinates position through NMEA protocol;
- **QRCode**: associates a matrix of bits to an URL. QRCodes are present in most of the shared means, semplifying the booking of that.

3.2 Functional requirements

3.2.1 Scenarios

Here are some scenarios that describe the usage of the system.

Scenario 1

Giovanni will start the fourth year of his Master's degree. Surfing the internet, he finds out that his lesson schedule for the first semester it has been published. Giovanni decides to fill in the application with his new appointments related to lessons attendance. In fact he knows where to go, at which time and day and for which amount of time. Since he knows that these events will going to happen for 3 months, he sets them as recurrent.

Scenario 2

da rileggere scritta veloce Giovanni want to start training but he doesn't know what are the best hours in which he can run in accord to his appointments, he know only that he can run between 5 and 7 pm, for 45 minutes. he can insert this last appointment in the application whitout specify the exactly starting hour and the system will give him the best hours in which he can run

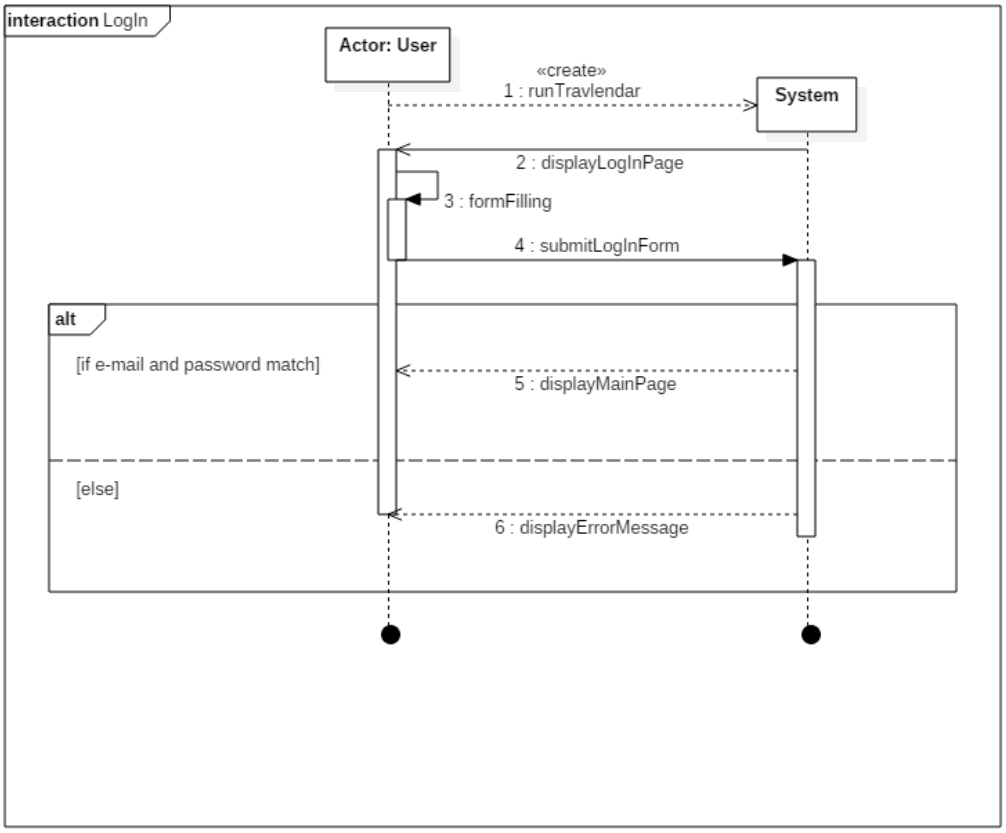
Scenario 3

da rileggere scritta veloce Giovanni has scheduled his appointments but at lunch time his son called him because he needed a ride for go back to home. Giovanni decided to help his son and so he brought him home. now the current running schedule is not more valid so he request to the system a reschedule of his appointment according to his position and the hour of day in which he is.

3.2.2 Use cases

User log-in

Name: User log-in
Actors: Registered User
Goals: (**Goal del login**
Input Condition: The user is registered to the system
Event Flow: 1. The user needs to log-in the application, so he runs it; 2. The system provides to the user a form to fill; 3. The user fills up the form with the his e-mail and his password (as said in 2.1.1) 4. The user submits the form to the system; 5. The system checks the user identity and provides to the user the main application page (**reference to the main application page**)
Output Condition: The user is logged-in to the system.
Exceptions: The user submits the form after having filled it with a wrong email or password.
Mapping on requirements: • Events from 3 through 5 granted by (**requirement che può recuperare informazioni dell'utente riguardo i dati della registrazione**); • Event 6 grandet by (**requirement che il sistema può controllare se l'identità di un utente è giusta**;



Appointment creation

Name: Appointment creation
Actors: Logged User
Goals: G1
Input Condition: • The user is registered to the system • The user is logged in to the systems

Event Flow:

1. The user wants to add a new appointment to his schedule;
2. The user requests the appointments page
3. The system provides the appointments page
4. The user requests the creation of a new appointment to the application;
5. The system provides to the user a form to fill;
6. The user fills up the form with the parameters (specified in 2.1.2) and constraints (specified in 2.1.4 about the new appointment;
7. The user submit the form to the system;
8. The system allocates the new appointment as Unscheduled (referring to statechart in figure;)
9. The system sends a confirmation to the user.

Output Condition: The user has created a new appointment;

Exceptions:

1. Some fields of the form referring to parameters are left blank;
2. The *location* field doesn't belong to the domain area of the application (**riferimento alla domain assumption della regione**)

Mapping on Requirements:

- Events 4 through 7 are granted by (**requirement che il sistema può recuperare informazioni riguardanti un appointment**)
- Event 7 is granted by (**requirement che il sistema è in grado di memorizzare un appointment**

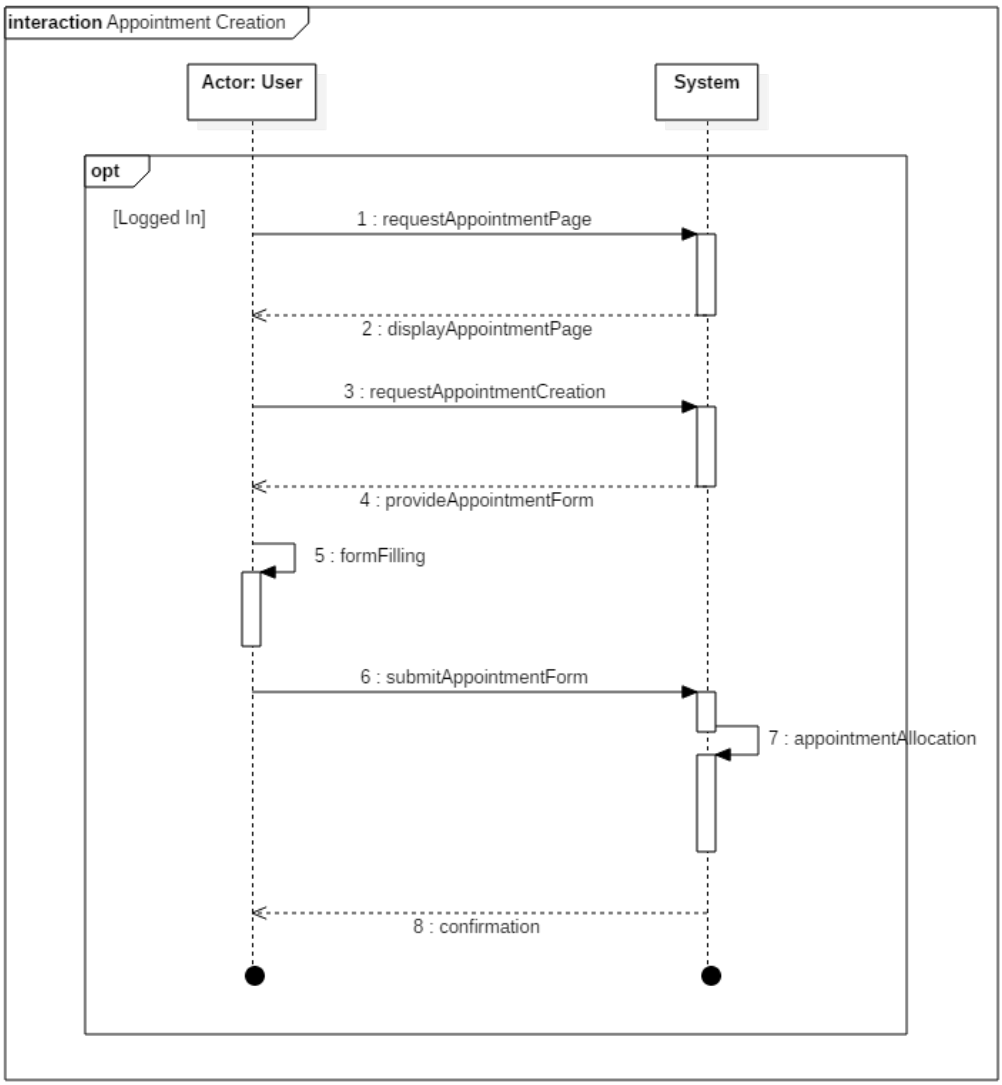


Figura 3.10: Appointment creation sequence diagram

In the sequence diagram there's the assumption that the log-in proceeds successfully. The log-in procedure referenced is the one explained in (**use case del log-in**.)

Schedule selection

Name: Multiple Schedules creation
Actors: Logged User
Goals: aggiungere rif al goal

Input Condition:

- The user is registered to the system
- The user is logged in to the systems

Event Flow:

1. The user wants to compare multiple schedules;
2. The user requests the schedules page;
3. The system provides the schedules page;
4. The user selects a schedule to be run;
5. The system display the mainpage with the schedule results (fare riferimento alla definizione)

Output Condition: The user selects a schedule to be run

Exceptions:

Mapping on Requirements:

- Events are granted by the requirment R10 **(mettere riferimenti)**

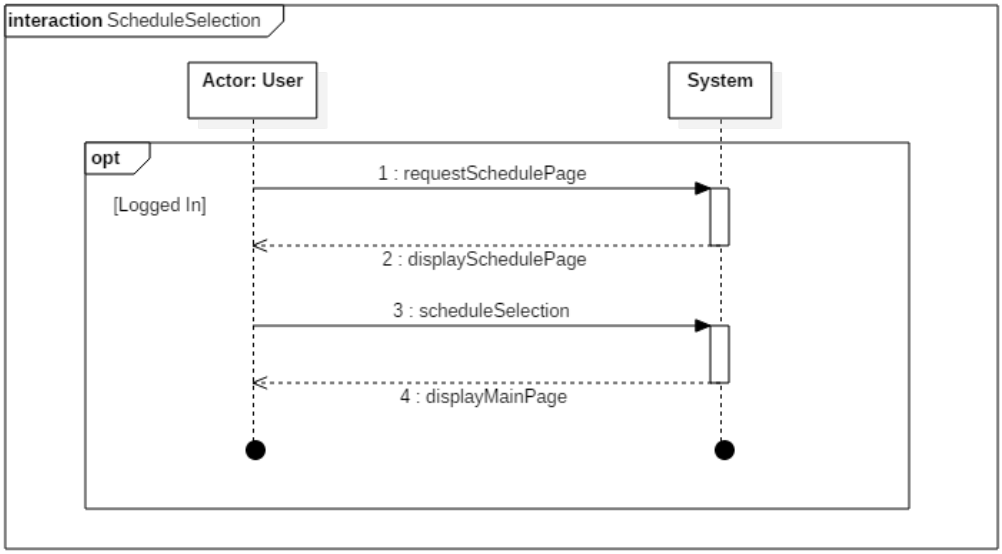


Figura 3.11: Appointment creation sequence diagram

Appointment editing

Name: Appointment editing
Actors: Logged User
Goals: (**goal che l'utente può modicare un appointment**)
Input Condition: The user is logged-in to the system

Event Flow:

1. The user wants to modify an appointment of his schedule;
2. The user selects the appointment to modify;
3. The system provides to the user the appointment form with all the parameters and constraints (with reference to and that were specified yet by the user;
4. The user edits the fields of the form;
5. The user submit the form to the system;
6. The system set the appointment as Unscheduled with the new parameters (referring to statechart in figure ..);)
7. The system sends a confirmation to the user.

Output Condition: The user has modified an appointment;

Exceptions:

1. Some fields of the form referring to parameters are left blank;
2. The *location* field doesn't belong to the domain area of the application (**riferimento alla domain assumption della regione**)

Mapping on Requirements:

- Events 3 through 5 are granted by **(the requirement that says that the system should let the user change the parameters and the constraints of an inserted appointment)**
- Event 6 is granted by (**requirement che il sistema è in grado di memorizzare un appointment modificato**

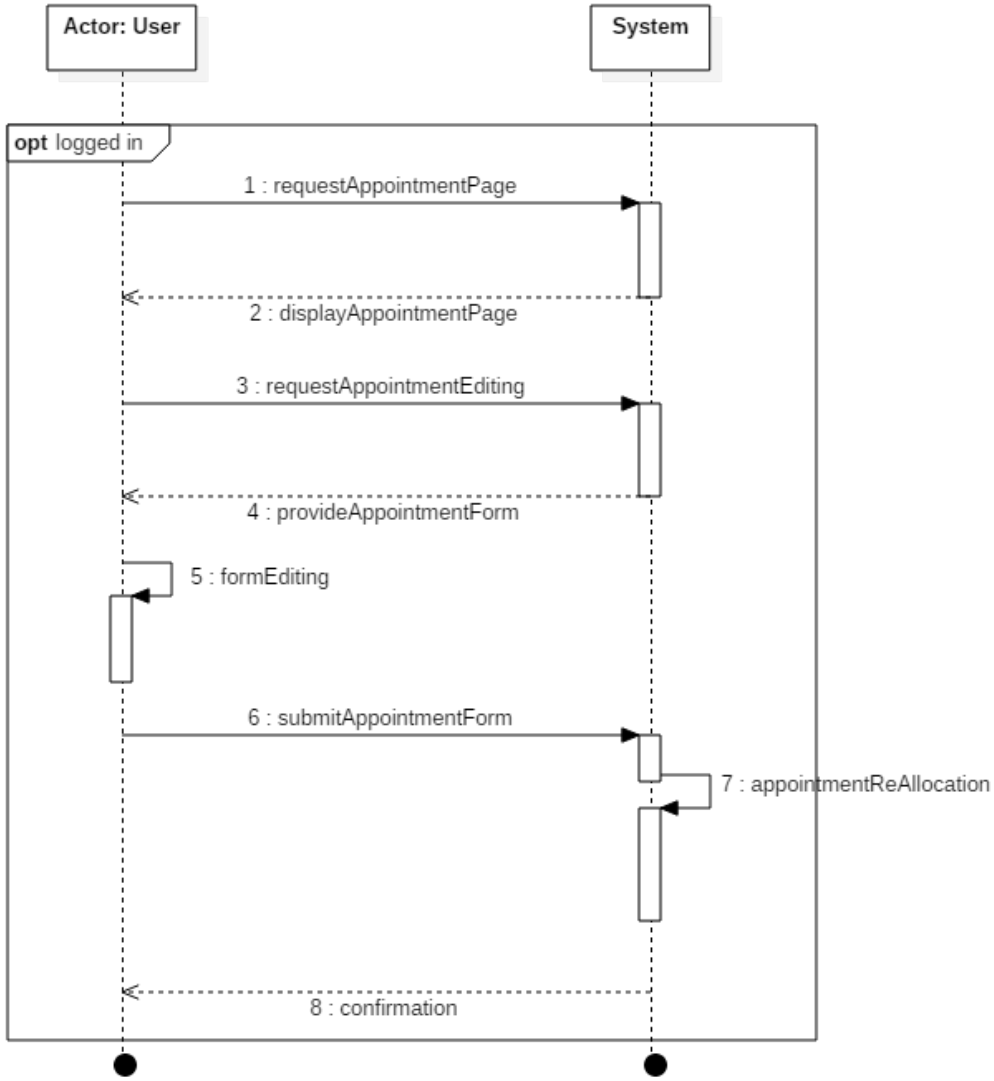


Figura 3.12: Appointment editing sequence diagram

Schedule appointments

Name: Schedule appointments
Actors: Logged User, External API
Goals: (**goal che permette di fare uno scheduling degli appuntamenti dell'utente**)
Input Condition: The user is logged-in to the system (in futuro mettere che l'actor è un logged user e togliere questo, se ci piace di più

Event Flow:

1. The user wants to schedule his appointments;
2. The user requests a schedule by mean of selecting the appointments to schedule of the current day;
3. The system provides to the user the schedule form with all the parameters, optimization criteria (with reference to (**parametri dello schedule** and **opt criteria dello schedule**) and the constraints (**riferimento ai constraints sullo schedule**.
4. The user fills up the fields of the form;
5. The user submits the form to the system;
6. The system retrieves information from external APIs about; travel options and related travel option data, weather forecast and strike days;
7. The system retrieves about the Travel Option Data of the newly created Schedule
8. The system stores the Schedule, together with his travel option data (**link alla definizione**) and stores it as Saved (**fare riferimento allo state chart**) with the appointment selected by the user.

Output Condition: The user has created a valid schedule of his appointments;

Exceptions:

1. Some fields of the form referring to schedule variables and optimization criteria are left blank. The parameters of schedule constraints could also be left blank since they will assume default values;
2. It's not possible to list the appointments as a Valid Schedule, so the schedule is Discarded **(riferimento allo statechart)**

Mapping on Requirements:

- Events 2 through 5 are granted by **(the requirement that says that the user can set the opt criteria, constraints, variables and of the schedules)**;
- Event 6 and 7 is granted by (**the requirement that says that we can retrieve info from APIs**);
- Event 8 is granted by (**the requirement that says that is possible to store a schedule**, (**requirement che il sistema è in grado di ottimizzare in base a quanto gli dico**)

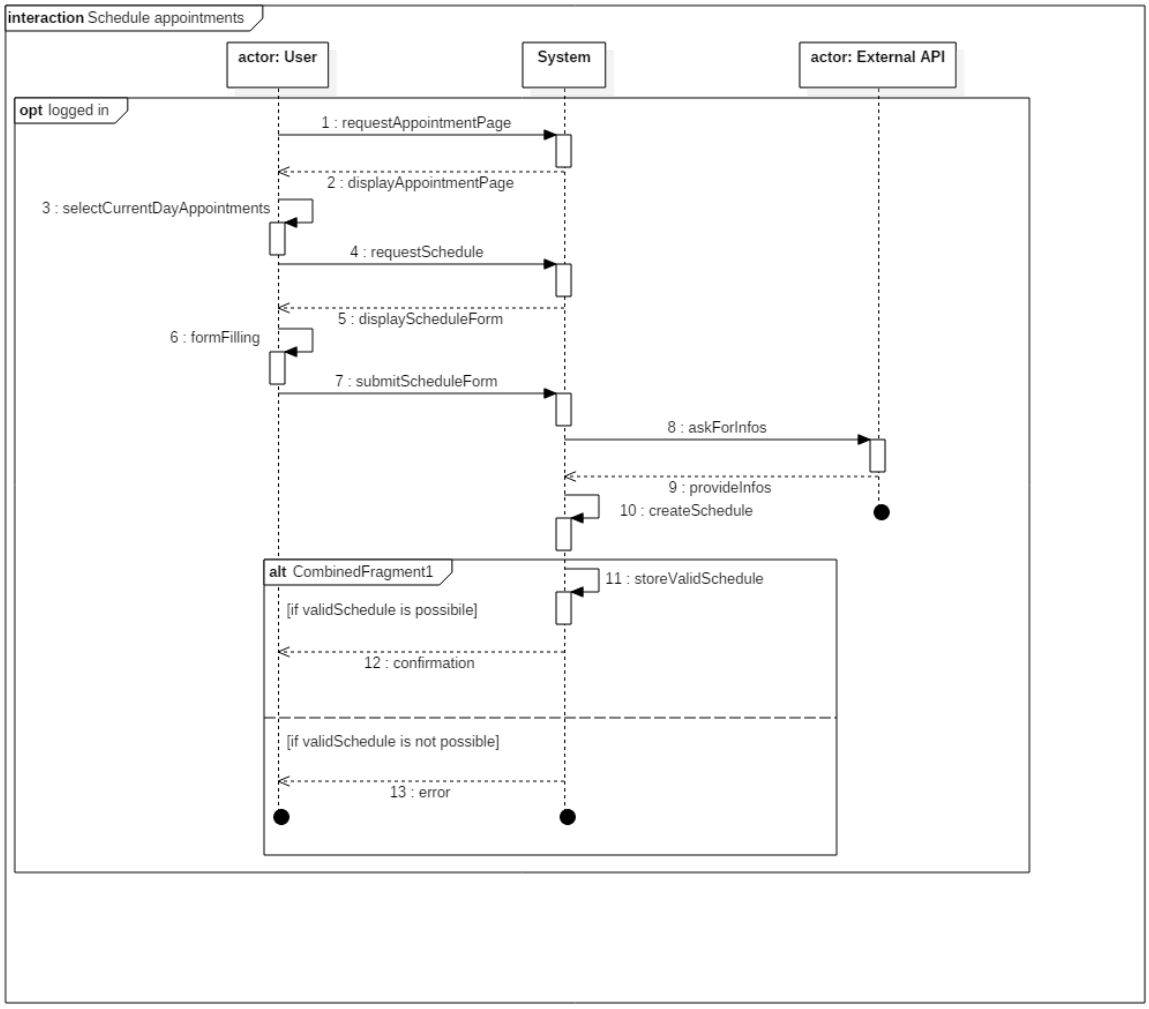


Figura 3.13: Schedule appointents sequence diagram

User registration

Name: User registration
Actors: External User, external e-mail service
Goals: G5
Input Condition:

Event Flow:

1. The user wants to register to the system, so he runs the application;
2. The system display the login/registration page
3. The user fills the form (the form is present on the first page that the application display after the startup)
4. The user submit the filled form to the system
5. The system send a confirmation e-mail to the user;
6. The system display a message in which the user is informed that he will recieve a confirmation e-mail;
7. The user confirm the registration by clicking a link in the received e-mail;
8. A confirmation message is sent to the application;
9. The user is redirected in his profile page inside the application;
10. The user specifies his parameters;

Output Condition: The registration is confirmed to the system;

Exceptions:

1. The e-mail given by the user is fake
2. The user makes a typo during the insertion of his e-mail

Mapping on Requirements:

- Events 1 through 3 are granted by R12;(AGGIUNGERE I RIFERIMENTI)
- Evens 4 through 6 are granted by R13. (AGGIUNGERE I RIFERIMENTI)

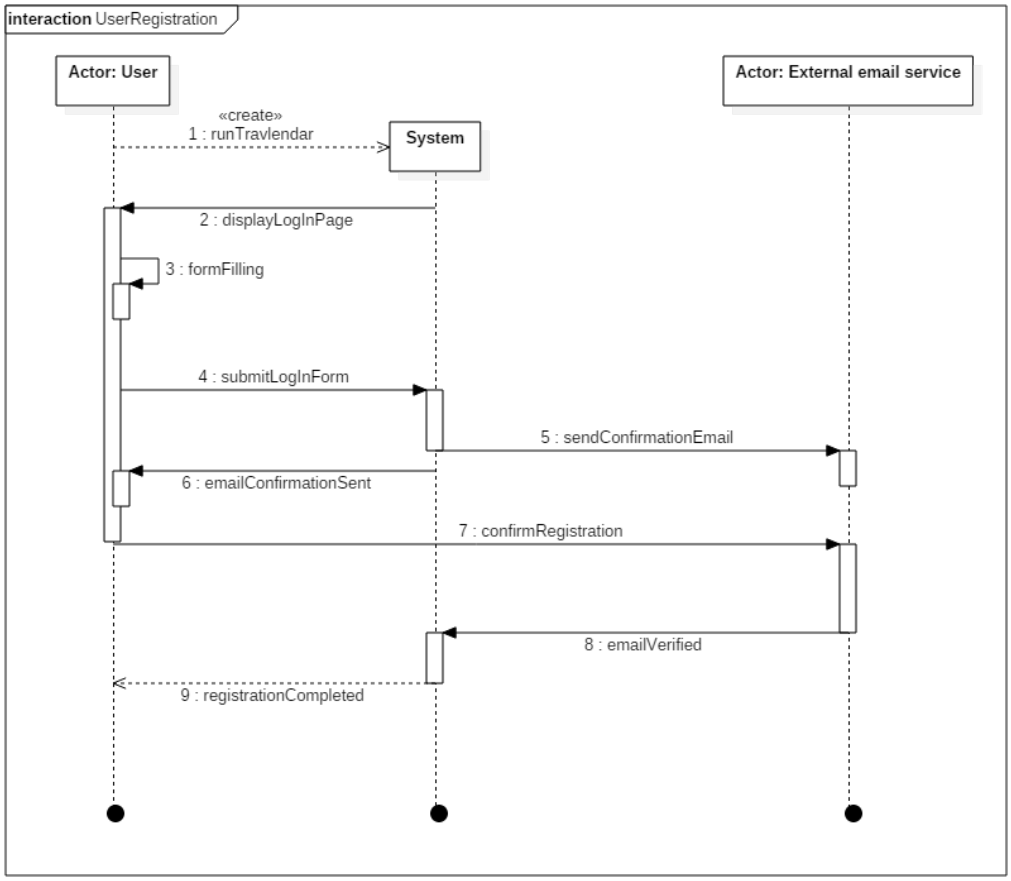


Figura 3.14: Registration sequence diagram

Booking phase

Name: Booking phase
Actors: Logged User, External APIs
Goals: FARE RIFERIMENTO AL GOAL CORRETTO

Input Condition:

- The user must be logged in to the system;
- The user must have selected a schedule to be run;
- The user must have linked to the system his external accounts;(FARE RIFERIMENTO ALLA SEZIONE DOVE SI PARLA DI QUESTA COSA)
- The user would like to buy the tickets for the travel means involved in the running schedule.

Event Flow:

1. The System, after the user have selected a schedule, asks to the user if he want to buy the ticket for the running schedule;
2. The User confirm to the system his intention;
3. The System perform a call to the travel means APIs for buying the ticket;
4. The APIs send back a confirmation message of the purchase;
5. The system send a confirmation message to the user.

Output Condition: The User recieve the confirmation message;

Exceptions:

1. The user doesn't have enough money in his card to complete the transaction;
2. there aren't free sits in one of the selected travel means;

Mapping on Requirements:

Events 3 through 5 granted by R17;(AGGIUNGERE I RIFERIMENTI)

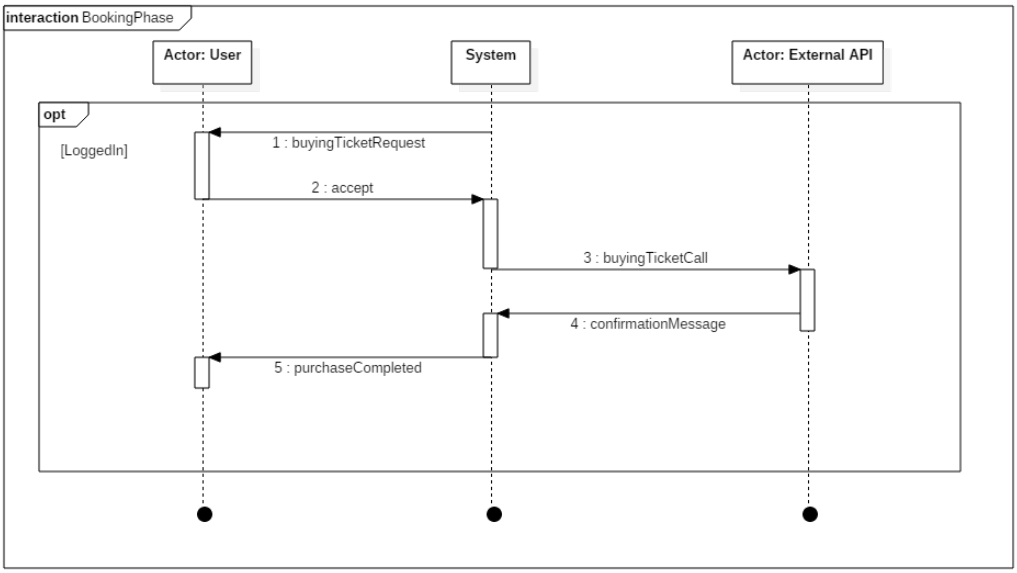


Figura 3.15: Booking phase sequence diagram

Dynamic directions

Name: Dynamic Directions
Actors: Logged User, External APIs, GPS
Goals: FARE RIFERIMENTO AL GOAL CORRETTO
Input Condition: • The user must be logged in to the system; • The user must have a running schedule;

Event Flow:

1. The user requests the Directions for the travel to the system;
2. The system retrieves the user position from his GPS;
3. The system retrives from external APIs the directions to give to the user based on his position;
4. The system display to the user the updated map and the directions that him must follow in order to arrive to the next appointment
5. user doesn't need more directions so he closes the dynamic map.

Output Condition: The User is satisfied with the information gathered until this moment so he decides to close the dynamic map;

Mapping on Requirements:

Events 3 through 5 granted by R17;(AGGIUNGERE I RIFERIMENTI)

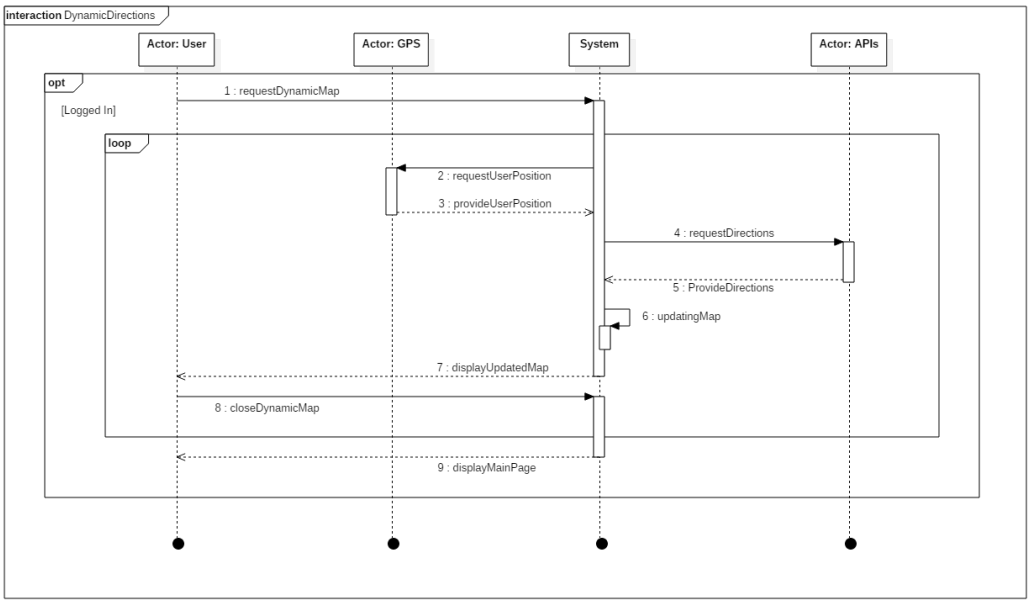


Figura 3.16: Booking phase sequence diagram

In this sequence diagram the loop of the actions described in the rectangle continues until the dynamicMap is closed (closeDynamicMap).

Recover credentials

Name: Schedule appointments
Actors: Registered User, External Email Service
Goals: (**goal che permette di recuperare le credenziali dell'utente**)
(in futuro mettere che l'actor è un registered user e togliere questo, se ci piace di più
Event Flow: 1. The user wants to recover his passoword; 2. The user requests to recover his passoword; 3. The system provides to the user the schedule form with his e-mail; 4. The user fills up the field of the form; 5. The user submits the form to the system; 6. The system sends the e-mail to the user with his password.
Output Condition: The user has recovered his password;
Exceptions: 1. The form it's left blank; 2. An invalid address is given;
Mapping on Requirements: • Actions 2 through 4 granted by requirement (**the system should be able to recover email address from the user** • Action 5 granted by the other requirement

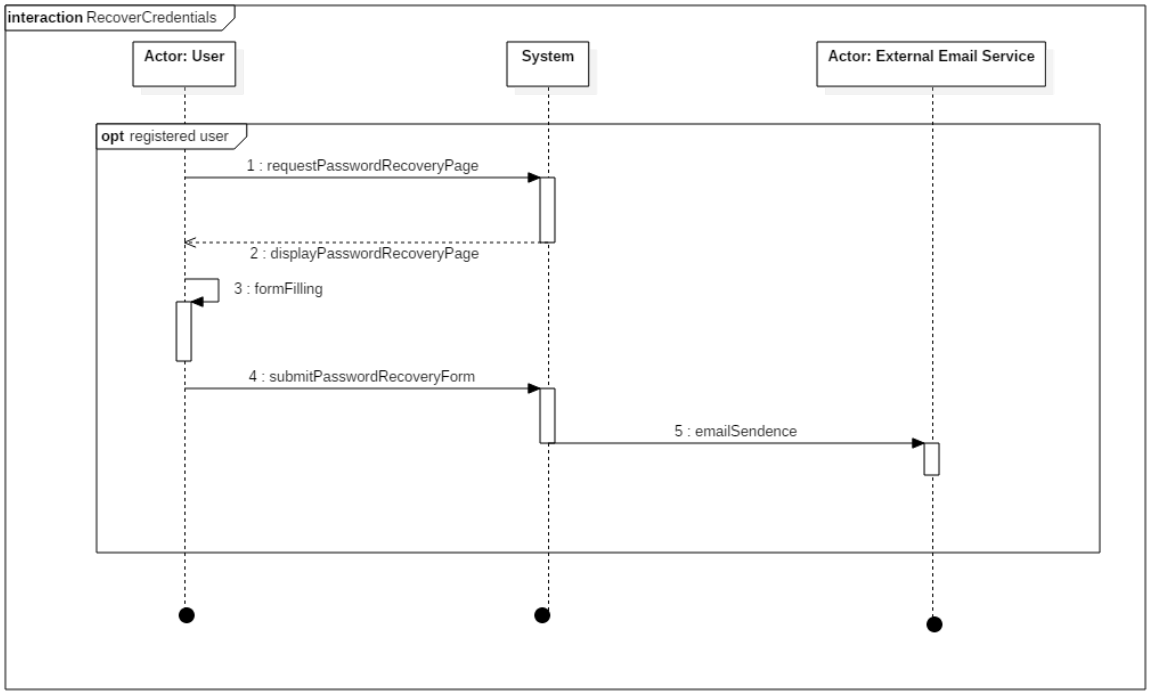


Figura 3.17: Password recovery sequence diagram

Notify Shared Means

Name: Notify Shared Means
Actors: Registered User, External Api
Goals: (**goal di notificare all'utente la possibilità di prendere i mezzi condivisi**)

Event Flow:

1. The system requests information to an external API about Shared Travel Means;
2. The external API service respond to the system with the information requested;
3. The system with the gathered information computes if there is a better path for the user according to the chosen constraints and optimization criteria;
4. if the path is found by the system is sent a notification to the user;

Output Condition: a better path is found;

Exceptions:

Mapping on Requirements:

- Actions 1 and 2 are granted by R28 **FARE RIFERIMENTO**
- Actions 4 is granted by R27 **FARE RIFERIMENTO**

AGGIUNGERE IMMAGINE GIA FATTA

3.2.3 Use Case Diagram

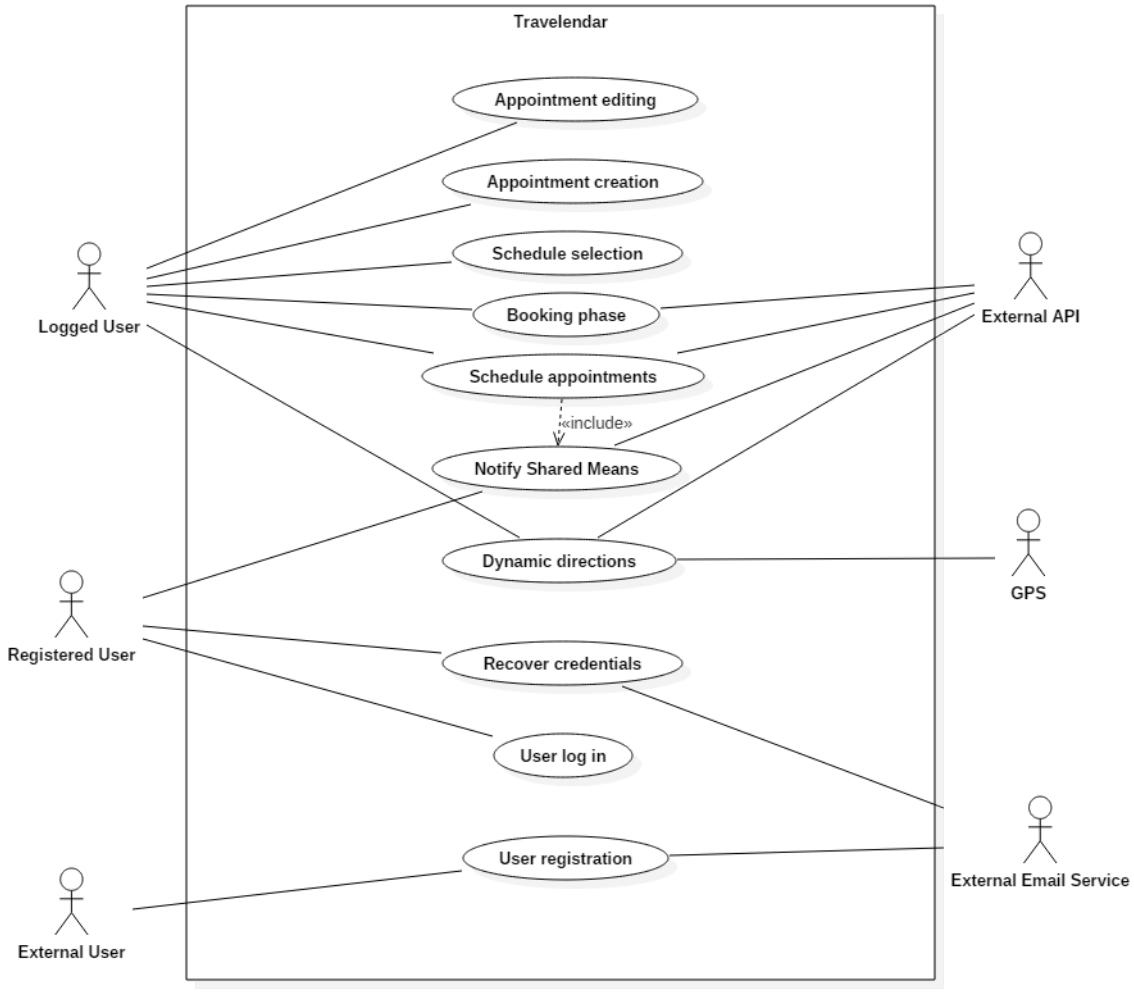


Figura 3.18: Use Case Diagram

3.2.4 Notes on diagrams

In these diagrams we assume that the user has ran the application. In particular, if the actor is a *Registered User* or an *Enternal User* then we assume that the user is facing the login page (**riferimento alla login page**. On the other hand, if the actor is a *Logged User* then is assumed that it is on the home page (**riferimento alla home page**.

3.3 Performance Requirements

The user must be notified in real time when a shared travel mean can be booked in order to provide a better mobility option, since these kind of transportation can remain available for a limited amount of time.

Other performance requirements can't be easily expressed because they depends heavily on external services and on the device in which the application is run. For example the time needed to create a schedule is influenced by the promptness of the APIs.

Anyway an upper bound of 5 seconds for the creation of a schedule is given. Moreover, the position of the user during the progress of a schedule must be track with a maximum delay of 100ms.

3.4 Design Constraints

3.4.1 Standard compliance

Our system conforms to OAuth2 [1] to handle the registration and login process. Moreover, HTTPS [2] protocol is used to guarantee secure treatment of user's sensitive data.

3.4.2 Hardware limitations

The bottleneck on the performance of the system is represented by the network infrastructure capability. In particular the most affected activity is the schedule computation.

Analysis

We can assume that the upload and download speeds of the APIs server is respectively 50Mbit/s and 150Mbit/s per client and that a request and a response weights are 20KB and 180KB. We can consider 2 cases:

- 10Mbit/s and 5Mbit/s download and upload speeds respectively of the client;
- 100Mbit/s and 50Mbit/s download and upload speeds respectively of the client.

[1] Open Authentication, an industry-standard protocol for authorization. It focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

[2] HTTPS is a communications protocol for secure communication over a computer network. The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.

No. Appointments	No. APIs calls	Request [MB]	Response [MB]	Tot [sec]
5 (2)	4	0.625	5.625	0.687
10 (4)	28	4.375	39.375	4.812
15 (7)	5047	788.6	7097.3	867.453

Tabella 3.10: Total times in the case of 10Mbit/s download speed and 5Mbit/s upload speed

No. Appointments	No. APIs calls	Request [MB]	Response [MB]	Tot [sec]
5 (2)	4	0.625	5.625	0.125
10 (4)	28	4.375	39.375	0.875
15 (7)	5047	788.6	7097.3	157.719

Tabella 3.11: Total times in the case of 100Mbit/s download speed and 50Mbit/s upload speed.

In the first column the numbers between brackets rempresent the number n of appointments with variable starting time, so that can be arranged differently relative to each other, changing their order in the schedule. Then the number of calls to External APIs that should be done is calculated, in case of a brute-force approach in the scheduling algorithm. Therefore the number of calls is proportional to $n!$. Finally the total amount of time is calculated considering the previous assumptions.

We can realize that the number of calls to external APIs should be minimized in order to fullfill the requirement on performance expressed in (**riferimento ai performance requirements**).

3.4.3 Any other constraint

3.5 Software System Attributes

3.5.1 Reliability

The system should guarantee that from the data retrieved is always constructed the most convenient valid schedules, according to user preferences and constraints, if it exists.

3.5.2 Availability

The system should be accessible 24 hours per day and should be available 99,9% of the time (up to 8,76 hours per year of downtime). Anyway the availability of the features

involving the use of external services can't be directly controlled. In particular the availability of the feature j is given by:

$$A_j = a_0 \prod_{i=1}^{n} a_i \tag{3.1}$$

where each a_i represent the availability of the external service i used and a_0 is the availability of the application. For instance, in the case of a schedule creation:

Figura 3.19: A failure in one of the chain of request to the APIs causes the entire process to break down

3.5.3 Security

The identity of the user must be verified through a login phase. User's characteristics must be protected during transmission from client to server throughout the registration. User credentials are cryptographied and then saved.

3.5.4 Maintainability

The system should be open to modifications. In particular the application should be able to consider new travel means, new scheduling optimization criteria and new constraints. Moreover, also the GUI should be easily editable, so that can adapt to new operating systems.

3.5.5 Portability

The system should be adaptable to run in all the devices (**link alla definizione**) considered.

Chapter 4

Alloy analysis

```
module Travlendar

open util/integer as Integer
open util/boolean as boolean
// impose ordering among relations
open util/ordering[Date] as DO
open util/ordering[Time] as TO
open util/ordering[ScheduledAppointment] as SAO

-----------------------------------------------
sig Date{}
-----------------------------------------------
sig Time{}

-----------------------------------------------
sig TimeSlot{
   start: one Time,
   end: one Time
}

-----------------------------------------------
sig OptimizingCriteria{}{
   this in Schedule.optimizingCriteria
}
```

```
-----------------------------------------------
sig User{
   hasCar: one Bool,
   hasBike: one Bool,
   hasPass: one Bool
}{
   this in Appointment.user
}

-----------------------------------------------
sig Schedule{
   date: one Date,
   wakeUpTime: one Time,
   optimizingCriteria: one OptimizingCriteria,
   constraints : set ConstraintOnSchedule,
   initialNumberOfPeopleInvolved : one Int
}

// there aren't schedules that doesn't have at least one ScheduledAppointment
fact NoScheduleUnlinked{
   all s:Schedule | s in ScheduledAppointment.schedule
}

// we can't have two different schedule for the same day with the same
    optimizing criteria
fact DifferentOptimizingCriteriaForScheduleOfTheSameDay{
   all s,s1 : Schedule| s != s1 and s.date = s1.date => s.optimizingCriteria
       != s1.optimizingCriteria
}

// no schedule of scheduled appointment belonging to different users
fact belongingCoeherence{
   no sa1,sa2 :ScheduledAppointment | sa1.schedule = sa2.schedule
   and sa1.appointment.user != sa2.appointment.user
}

-----------------------------------------------
```

```
sig Appointment{
   user: one User,
   date: one Date,
   startingTime: lone Time,
   timeSlot: lone TimeSlot,
   constraints: set ConstraintOnAppointment,
   variationNumberInvolvedPeople: one Int
}
{
   // startingTime and timeSlot are mutually exclusive
   startingTime = none => timeSlot != none
   timeSlot = none => startingTime != none
   startingTime in TO/nexts[timeSlot.start] and startingTime in
       TO/prevs[timeSlot.end]
}

// all appointments must be scheduled into some ScheduledAppointment
fact AppointmentAssociationCoherence {
   all a : Appointment | a in ScheduledAppointment.appointment
}

-----------------------------------------------
sig ScheduledAppointment{
   schedule: one Schedule,
   date: one Date,
   appointment: one Appointment,
   startingTravelTime: one Time,
   ETA: one Time,
   endingTime: one Time,
   numberOfInvolvedPeople: one Int,
   weather: one Weather
}
{  startingTravelTime in TO/prevs[endingTime]
   // ETA must be between starting and ending times
   ETA in TO/nexts[startingTravelTime] and ETA in TO/prevs[endingTime]
   date = schedule.date
   date = appointment.date
```

```
   numberOfInvolvedPeople >= 0
   startingTravelTime in TO/nexts[schedule.wakeUpTime]
}

// appointments starting time ordering coherence
fact ScheduledAppointmentsOrderingConsistence {
   all a1,a2 : ScheduledAppointment | a1.schedule = a2.schedule and a1 in
       SAO/prevs[a2]
   => a1.startingTravelTime in TO/prevs[a2.startingTravelTime]
}

// if two scheduled appointments are relative to the same appointment then
    they must
//belong to different schedules
fact AppointmentOnMultipleSchedules {
   all s1,s2 : ScheduledAppointment | s1 != s2 and s1.appointment =
       s2.appointment
   => s1.schedule != s2.schedule
}

// Starting time of a scheduled appointment must coincide with the starting
    time of the
//non-scheduled appointment if timeSlot is not specified
fact StartingTimeCoherence{
   all s : ScheduledAppointment | s.appointment.timeSlot = none =>
   s.startingTravelTime = s.appointment.startingTime
}

// The number of involved people in each travel is coherent with the number of
    seats of each
//travel mean used for it
fact numberOfPeopleInvolvedCoherentWithSeats{
   no p : Path | p.travelMean.seats < p.source.numberOfInvolvedPeople
}

// The selected travel mean for a path can provide enough seats for the people
    involved in the appointment
```

```
fact coherenceOnNumberOfInvolvedPeople{
   all sa:ScheduledAppointment, a:Appointment, s:Schedule |
   sa in schedule.s and a in sa.appointment =>
   sa.numberOfInvolvedPeople = add[s.initialNumberOfPeopleInvolved,
      sum e : SAO/prevs[sa] | e.appointment.variationNumberInvolvedPeople]
}

-------------------------------------------------
sig Path{
   lenght: one Int,
   source: one ScheduledAppointment,
   dest: one ScheduledAppointment,
   travelMean: one TravelMean
}
{
   lenght >= 0
   source != dest
   source.schedule = dest.schedule
}

// No path linking to scheduled appointment belonging to different user
fact SchedulePathBelongingConsistency{
   all p:Path | p.source.appointment.user = p.dest.appointment.user
}

// No scheduled appointment unreachable
fact AllScheduledAppointmentCanBeReachedByPaths{
   all s : ScheduledAppointment | s in (Path.source + Path.dest)
}

// No path belonging to a user with a travel mean that he/she doesn't own
fact TravelMeanConsistency{
   no p:Path| (p.travelMean = Bike and p.source.appointment.user.hasBike in
       False) or
   (p.travelMean = Car and p.source.appointment.user.hasCar in False)
}
```

```
// No path linking two scheduled appointments that are not sequential
fact PathOrderConsistency{
   all p:Path | p.source = SAO/prev[p.dest]
}

-------------------------------------------------
abstract sig Constraint{
   travelMean: one TravelMean,
   maxTravelDistance: one Int // with travel mean above
}
{maxTravelDistance >= 0} //if 0 implies that the travel mean is deactivated

sig ConstraintOnAppointment extends Constraint{
}
{maxTravelDistance = 0}

sig ConstraintOnSchedule extends Constraint{
   weather: set Weather,
   timeSlot: lone TimeSlot,
   strikeDate: lone Bool
   // in which travel mean can't be used
}{
   weather != none => maxTravelDistance = 0
   weather != none => (timeSlot .start = timeSlot.end)
}

fact NoConstraintUnlinked{
   all c : Constraint | c in Appointment.constraints or c in
       Schedule.constraints
}

-----------------------------------------------
abstract sig Weather{}

one sig Sunny extends Weather{}
one sig Snowy extends Weather{}
one sig Rainy extends Weather{}
```

```
one sig Foggy extends Weather{}
one sig Cloudy extends Weather{}

fact NoWeatherUnlinked {
   all w : Weather | w in ConstraintOnSchedule.weather
}

-----------------------------------------------
abstract sig TravelMean{
   seats: one Int
}
//100 seats encode an unbounded number of seats

abstract sig PublicTravelMean extends TravelMean{}
abstract sig SharedTravelMean extends TravelMean{}
abstract sig PrivateTravelMean extends TravelMean{}

sig Train extends PublicTravelMean{}{
   seats=100
}
sig Bus extends PublicTravelMean{}{
   seats=100
}
sig Tram extends PublicTravelMean{}{
   seats=100
}
sig Taxi extends PublicTravelMean{}{
   seats=3
}
sig Underground extends PublicTravelMean{}{
   seats=100
}
sig CarSharing extends SharedTravelMean{}{
   seats=5
}
sig BikeSharing extends SharedTravelMean{}{
   seats=1
```

```
}
sig Bike extends PrivateTravelMean{}{
   seats=1
}
sig Car extends PrivateTravelMean{}{
   seats=5
}
sig Walking extends PrivateTravelMean{}{
   seats=100
}


-----------------------------------------------
// if a schedule appointment is on a schedule, it must be in the list of
    scheduled appointments of that schedule
ScheduledAppointmentAndScheduleBiunivocity : check{
   all a : ScheduledAppointment | all s : Schedule | a.schedule = s => a in
       schedule.s
} for 8
-----------------------------------------------------
// if a scheduled appointment is in a schedule, they must have the same date
EveryAppoinmentOfScheduleIsInItsDay : check {
   all s : Schedule, a : ScheduledAppointment | a.schedule = s => s.date =
       a.date
}
for 8
-----------------------------------------------------
ScheduleIsOwnedByOnlyOneUser : check {
   all s1,s2:ScheduledAppointment | s1.schedule = s2.schedule implies
       s1.appointment.user = s2.appointment.user
} for 8
-----------------------------------------------------
// all scheduled appointment date must be equal to its original appointment
    date
AppointmentDateIsEqualToItsScheduledAppt : check {
   all a : ScheduledAppointment | a.date = a.appointment.date
} for 8
-----------------------------------------------------
```

```
//source and dest of each path must refer to the same schedule
sourceDestOfPathCoherence : check {
   all p:Path | p.dest.schedule = p.source.schedule
} for 8
-----------------------------------------------------
// scheduled appointments are ordered according to their path precedence
PathOrderConsistency : check {
   all p:Path | p.dest = SAO/next[p.source]
} for 8
-----------------------------------------------------
/* if two appointments belong to different user then they should belong to
    different
    schedule since a schedule belongs to only one user */
TwoAppointmentsWithSameUserNoInSameSchedule : check {
   all a,a1:Appointment | a.user != a1.user => appointment.a.schedule !=
       appointment.a1.schedule
} for 8
-----------------------------------------------------

// return all the paths of the specified schedule
fun pathOfSchedule [s : Schedule] : set Path {
   (source+dest).(schedule.s)
}

// return the total distance travelled in the specified schedule with a
    specific travel mean
fun distanceTravelledWithMeanInSchedule [s : Schedule, t: TravelMean] : Int {
   sum e : (pathOfSchedule[s]) & (travelMean.t) | e.lenght
}

// checks if the specified path links two scheduled appointment in the given
    schedule
pred doesPathBelongToSchedule [s:Schedule, p:Path] {
   some sa:ScheduledAppointment | sa in (Path.source + Path.dest) and
       sa.schedule=s
}
```

```
// checks if a schedule satisfying the specified constraint exists
pred doesConstraintSatisfySchedule (c : ConstraintOnSchedule){
   some s : Schedule | all p:Path |
   doesPathBelongToSchedule[s, p] and
   p.travelMean != c.travelMean or (p.travelMean = c.travelMean
              and (schedule.s).weather not in c.weather
              and ((c.timeSlot.start = c.timeSlot.end)
                 or ((schedule.s).ETA in TO/prevs[c.timeSlot.start]
                 or (schedule.s).startingTravelTime in TO/nexts[c.timeSlot.end]
                     ))
              and (c.strikeDate in True implies p.travelMean not in
                  PublicTravelMean)
              and (distanceTravelledWithMeanInSchedule[s,c.travelMean] <
                  c.maxTravelDistance )
           )
}

// checks if a scheduled appointment satisfying the specified constraint exists
pred doesConstraintSatisfiesAppointment(c : ConstraintOnAppointment){
   some s: Schedule | all sa:ScheduledAppointment, a: Appointment, p: Path |
      sa in schedule.s and sa.appointment = a and p.dest=sa
      => p.travelMean != c.travelMean
}

pred NoOverlappingScheduledAppointmentInSchedule (s : Schedule) {
   all s1,s2 : ScheduledAppointment | s1.schedule = s and s2.schedule = s and
       s1 != s2
   => s1.endingTime in TO/prevs[s2.startingTravelTime] - s2.startingTravelTime
   or s2.endingTime in TO/prevs[s1.startingTravelTime] - s1.startingTravelTime
}

// checks if a valid schedule exists (see def. on RASD document)
pred validSchedule {
   some s: Schedule | all cs:ConstraintOnSchedule, ca:ConstraintOnAppointment |
   cs in s.constraints and ca in (schedule.s).appointment.constraints
   and doesConstraintSatisfySchedule[cs] and
       doesConstraintSatisfiesAppointment[ca]
```

```
   and NoOverlappingScheduledAppointmentInSchedule[s]
}
-------------------------------------------------

// new appointment insertion action
pred InsertNewAppointment[u,u1 : User, a1 : Appointment]{
   //precondition
   all a : Appointment | a.user = u and a1.user != a.user
   //postcondition
   user.u1 = user.u + a1
}

// checks if the insertion of a new appointment is correct
InsertNewAppointmentIsCorrect : check {
   all u,u1 : User, a,a1 : Appointment | a.user = u and a1.user != a.user and
       InsertNewAppointment[u,u1,a1]
   => a1 not in user.u and a1 in user.u1
} for 8
-------------------------------------------------

// editing appointment action
pred ModifyAppointment[a,a1 : Appointment, u : User]{
   //precondition
   a in user.u and a1 not in user.u
   //postcondition
   a1 in user.u and a not in user.u
}

// checks if the modification of a new appointment is correct
ModifyAppointmentIsCorrect : check {
   all u : User, a,a1 : Appointment | a in user.u and a1 not in user.u and
       ModifyAppointment[a,a1,u]
   => a1 in user.u and a not in user.u
} for 8


pred show(){}
```

```
run { show and validSchedule} for 7 but 4 Int
```