Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Progetto di Ingegneria del Software 2

Parte I: RASD

Prof.ssa Di Nitto
A.A. 2017/18

Autori	Matr.
Amadelli Federico	899367
Artoni Alessandro	899343
Baccelli Alessio	898514

Requirements
Analysis
Specification
Document

Progetto *Travlendar+*

Chapter 1

1. Introduction

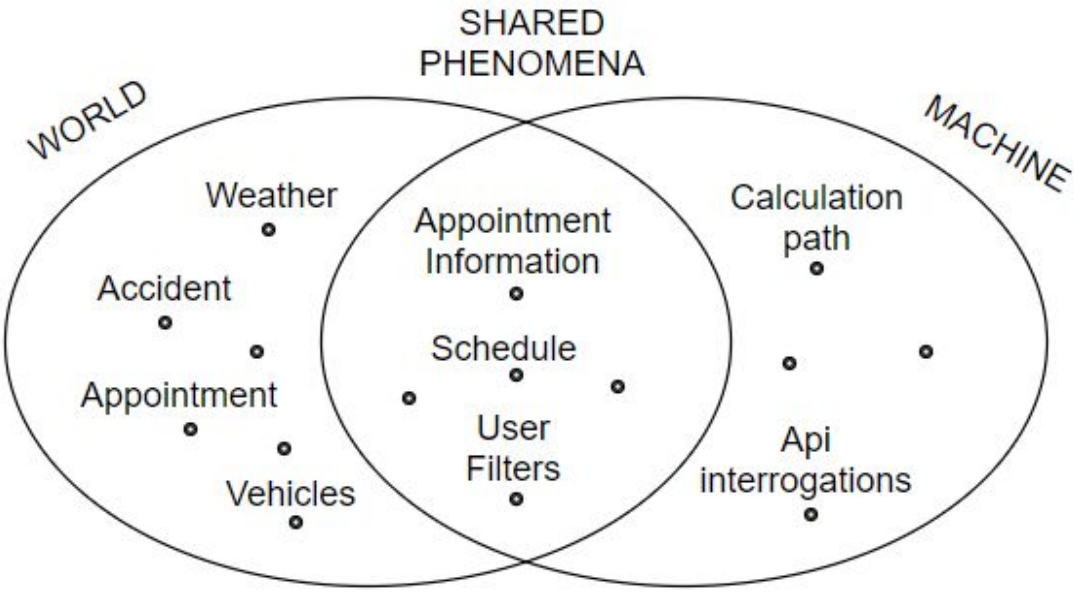
Travelling has never been so easy. Not only traditional vehicles improved, but new phenomena like car and bike sharing appeared. Mobility itself is changing.
People started using new services to move, and when they need to visit a new place often rely completely on Google Maps. The same users need a better manage of their schedules.
Therefore this project wants to help users to schedule their meetings accounting the travel time between each appointment giving the possibility to customize each appointment and each travel path.

1.1. Purpose

This document is the Requirement Analysis and Specification Document (RASD) for the Travlendar+ applications. The aims of this document are to describe the system in terms of functional and nonfunctional requirements, analyze the real needs of the customer in order to model the system, show the constraints and the limits of the software and indicate the typical use cases that will occur after the release.
This document is addressed to the developers, testers and software designers who have to implement the requirements.

1.2. Scope

The system, in order to achieve all the aforementioned goals has to interact both with users and other external services (like weather forecasting, ticket prices, etc.) which will help the application in giving a complete and reliable service.



This is an example of how the machine interacts with the environment.
Users insert their appointments and the machine calculates the best path, according to user's preferences, making interrogations to an appropriate API, taking into account the possibility of accidents and the weather.

1.2.2 Goals

This project wants to:

[GOAL0] Allow users to login and register into the Web Application;

[GOAL1.0] Allow users to build a calendar where they can add appointments;

[GOAL1.1] Give the possibility to schedule particular appointment

[GOAL1.2] Set a "Level of importance" of a certain appointment.

[GOAL2.0] Identify a best path solution in order to reach one appointment from another;

[GOAL2.1] Allow clients to filter the travel solution to select the one they prefer;

[GOAL2.2] Show all the possible travel solutions to arrive at the next appointment

[GOAL3.0] Support users in their travel helping them buying tickets.

[GOAL3.1] Allow user to add a pass from the Web Application

[GOAL4.0] Support users in their everyday life trouble

[GOAL4.1] Tell users what the weather is in a certain day.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definition

- *Vehicles.* All the means of transport (bike, bus, car...) that the user can use to reach the point of interest.
- *Appointment***.** An arrangement to meet someone at a particular time and place, it comprends working appointments, dates, all the events planned by the user.
- *Schedules.* A plan for carrying out a lists of intended events and times.
- *Travel path.* Is the roadway chosen by the system based on the point the user has to go.
- *Accident.* All the event that can happen to delay the user during the travel path.
- *User filters.* The user's preferences on the travel path, like elements to avoid, or elements to take.
- *Id*: identification of the user to access the application
- *Password*: a secret string used by the user to verify his own account
- *Urgency level:* for each appointment is possible to set up a different level, called urgency level, it gives the appointment a certain level of importance, higher is the lever, more important is the appointment, lower is the delay margin.
- *Actor*: an entity that interacts with the system
- *Break:* all types of breaks the user might select to have during the day (e.g. lunch). All breaks are intended to be "flexible": the user has the opportunity to choose a time slot and the duration of the break. The app will then schedule the user's appointments during the day in order to reserve the amount of time specified for the break in the indicated time slot.
- *User:* in our case, when talking about users we're just considering the costumer of our application. This means it is not a two-sided platform application.

1.3.2. Acronyms

- API: Application Programming Interface.
- DBMS: Database Management System.
- UI: User Interface.
- RASD: Requirement Analysis and Specification Document.
- ETA: Estimated Time of Arrival.
- MTBF: Mean time between failures.
- MTTR: Mean time to repair.
- UL: urgency level. Is the importance level that the user set to the appointment.

1.3.3. Abbreviations

- Req. as for Requirement.
- Dom. as for Domain assumption.
- WebApp as for Web Application.
- App for Application
- Dep. as for Dependency

1.4. Revision history

Version	*Date*	*Authors*	*Summary*
1.0	/	Amadelli & Artoni & Baccelli	Release
1.1	22/11/2017	"	Modified Goals, req, Dom
1.2	26/12/2017	"	Updated UML. Other minor changes.
1.2.1	04/01/2018	"	Added [DOM. 4.3]
1.2.2	10/01/2018	"	Minor changes

1.5. Reference Documents

1.5.1. Google's API

The web application will use the javascript Google APIs.
https://developers.google.com/maps/documentation/javascript/reference#DirectionsService
The mobile application will use the android Google APIs.
https://developers.google.com/android/reference/com/google/android/gms/maps/package-summary .

1.5.2. Carbon Footprint references

All vehicles carbon footprint are based on values coming from:

- Department for Environment, Food and Rural Affairs (DEFRA-UK)
- Vehicle Certification Agency (VCA) - UK
- World Resource Institute (WRI), Greenhouse Gas (GHG) Protocol

1.5.3. Weather APIs

The system will use weather API.
https://openweathermap.org/api

1.6. Document Structure

The first chapter is an introductory chapter. It defines the purpose and the scope of our application, gives references to terms, definitions and acronyms and to the APIs we used.
The second chapter is an overall description of our application.
In the third, there's a glimpse of the application's interfaces and are presented all the standard attributes in order of reliability, security, etc..
In the fourth we start making a formal analysis of the system. Scenarios, use cases, bpmn, class diagram, alloy's analysis are presented to the reader.
In the last chapters we wrote how many hours each team member spent writing the RASD and the tool we used.

Chapter 2 - Overall Description

2.1. Product perspective

The product is a web based system implementing a client-server model.
The web-access and the application will be free-access. Still the source code will be private.
Travlendar+ provides simple mechanism for users to schedule appointments and organize travels betweens the formers.
It will be a stand-alone product relying on different external information.

2.2. Product functions

Travlendar+ is created to solve all schedule problems a person can meet.
It helps creating a calendar, managing meetings and appointments either these are job-meeting or something else like lunch or normal activities.
In order to do that, it will help users selecting their best travel option, either it will be the lowest footprint or the fastest option, taking into account also the weather, the traffic and all the near bikes and cars of the various bike and car sharing. More details on the functionality of the product will be discussed in section [3.2].

2.3. User Characteristics

It is considered that the user has an internet access and a basic knowledge of operating on digital calendars.
Our target are all of these people who require a basic management of their schedules but can't afford to hire a secretary. Nevertheless, the system can help secretaries to manage their boss meetings.

2.4. Assumptions, dependencies and constraints

[DEP1] The possibility to schedule appointments and see the existing agenda is under the responsibility of the service and it is strictly correlated with his running. However, the system uses external API's, for example in order to compute the shortest path. Therefore, the continuation of the service cannot be ensured if errors or problems occur to these external platforms.

[DEP2] The punctuality of all the public transportation cannot be guaranteed as some unpredictable accidents may happen. The time table used by the system is the official one, provided by the managing institution of the public transportation.

[DEP3] When the user selects the lowest footprint, the system will act using an average stored value, linking each vehicle to a particular value.
These values are taken by sites and external information which we therefore consider to be accurate. [1.5.2]

[DOM1.0] All the new events inserted by the user are added after the current date

[DOM1.1] The system allows the user to add overlapped events. For those, he won't calculate the ETA. Still, a path can be retrieved.

[DOM1.2] Breaks can be in one place (eg. in a park) or in none.

[DOM2.0] No GPS is used to determine the current position

[DOM2.1] Because of [DOM2.0], the information gave by the travelling systems concerns only pre-trip support in order to accomplish what user asks. Further information on any kind of delays won't be retrieve to the user

[DOM2.2] When using external APIs their information will have an amount of precision sufficient for our needs.

[DOM2.3] ETA is shown in hours/minutes format.

[DOM2.4] For the moment only few filters are available

[DOM3.0] All public transportation are considered to be in Lombardia

[DOM3.1] System does not check if the user does indeed have the pass. It will assume that the user owns it.

[DOM4.0] Once a strike is declared, we assume it is not revoked. Moreover, there's in general a risk of unavailability (not a certainty) of some travel means.

[DOM4.1] To notify the system, the user must have been close to the reported zone

[DOM4.2] Weather will be precise

[DOM4.3] There won't be more than 3 events per minute.

[DOM4.4] We assume the existence of a 3rd party API that tells us when there's a strike

Chapter 3 - Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

This is a mockup of the user interfaces of the web application seen through a mobile device.

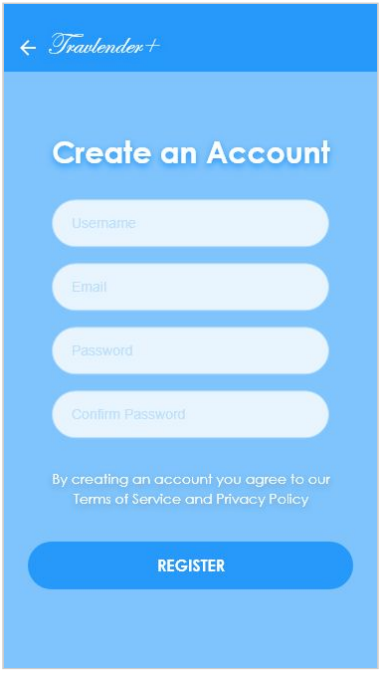


Figure *3.1.1.1*
User here can register

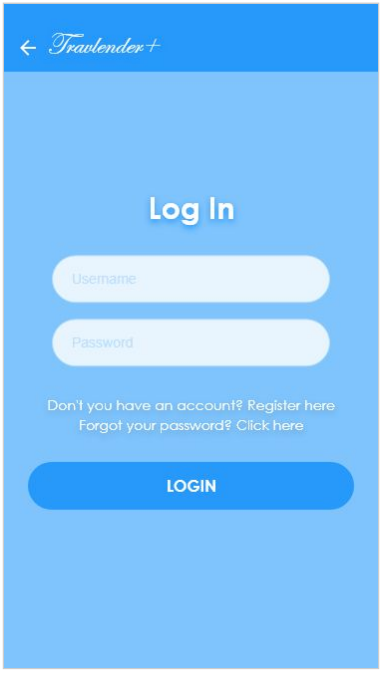


Figure *3.1.1.2*
User here can login

Figure *3.1.1.3*
In this figure the user has the first overview of the application.

In figure [3.1.1.3], user sees the main application page. The icons on the right represent the weather of that day. If it rains, the application suggests to take the umbrella.
By selecting a day, the user is moved to figure [3.1.1.5] where he can add an event clicking the + button.

Figure *3.1.1.4*

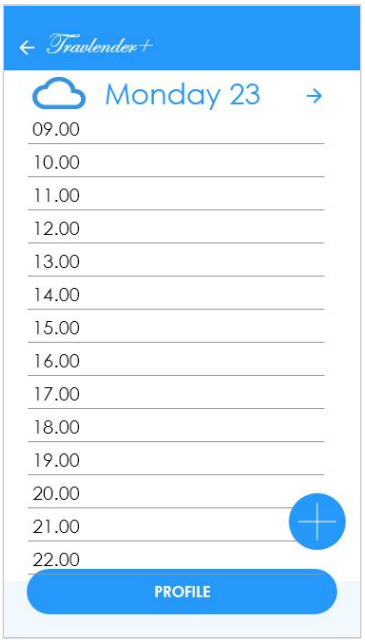


Figure *3.1.1.5*

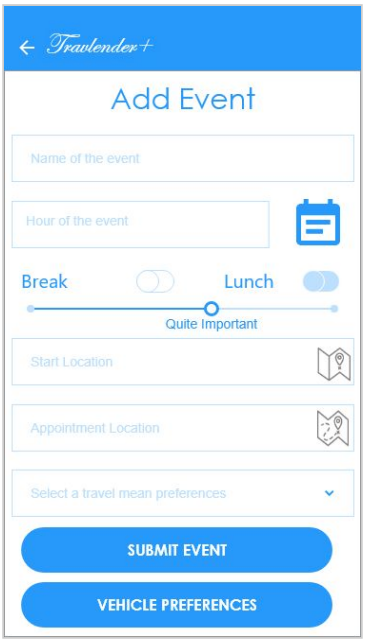


Figure *3.1.1.6*

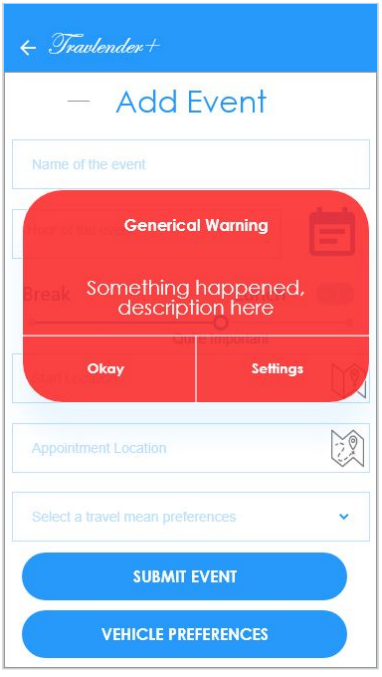


Figure *3.1.1.7*
Example of a general warning.

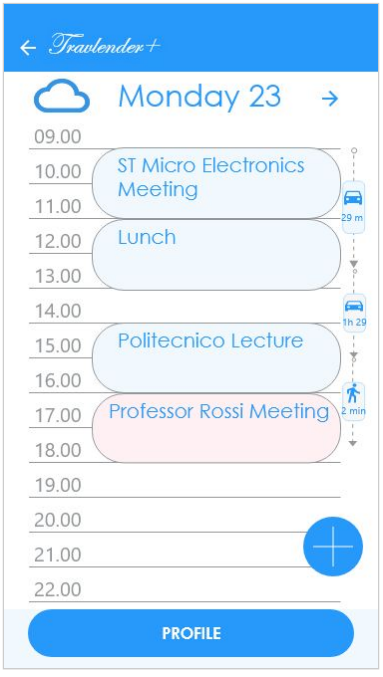


Figure *3.1.1.8*
Example on how application will show appointments

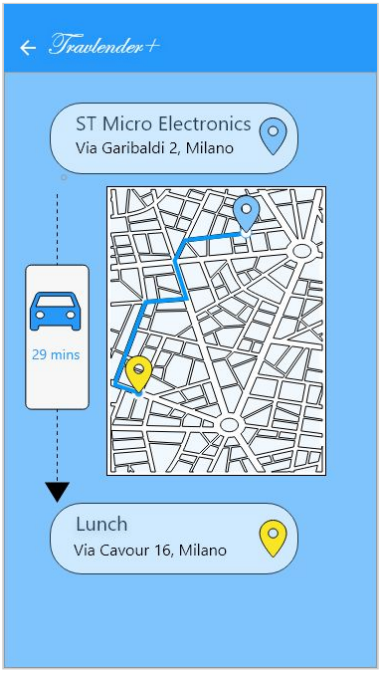


Figure *3.1.1.9*
Travel path of a particular vehicle to reach an appointment.

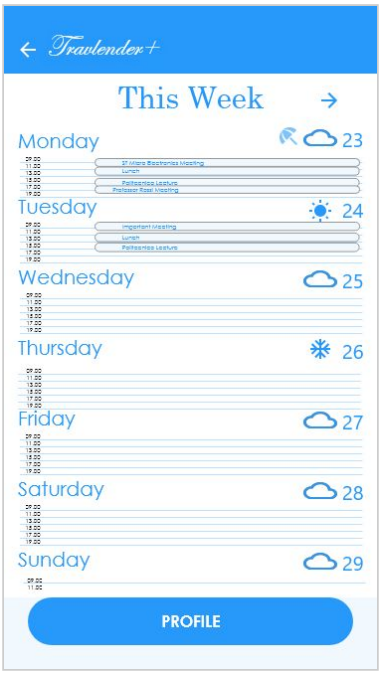


Figure *3.1.1.10*

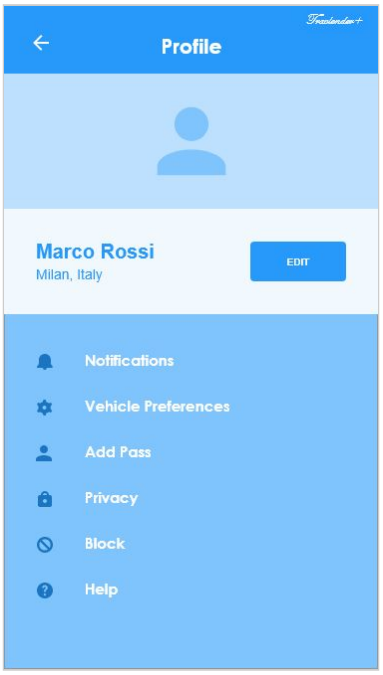


Figure *3.1.1.11*
User's profile.

Figure *3.1.1.12*
User's vehicle preference

3.1.2. Hardware Interfaces

There won't be any Hardware interfaces. Users will be able to connect to Travlender+ using the application or the website.

3.1.3. Software Interfaces

Travlendar+ will use a Google's API, Google Maps, which will be used in order to compute the shortest travel path between the starting point and the arrival. Moreover, it will also compute the travel distance and the ETA taking into account the traffic and the type of the vehicles used. Also, another API will be used in order to show to the user the location of the bikes and cars of the bike and car sharing system.
As more users will join Travlendar+, expanding towards different regions has to add other vehicles (like trains or Airplanes) and this will require other APIs (like Trenitalia).

3.1.4. Communication Interfaces

There aren't any direct communication functions related to this product like form requests or e-mails communication.
Anyway, since the application will communicate to a server, we will adopt a REST architecture using http over transport layer security as a communication standard, in order to protect against man-in-the-middle attacks.

3.2. Functional Requirements

3.2.0. [G0] Major Function of the Login and Registration System

[REQ0] Authenticate and login user to the application if the password he inserts matches.

[REQ1] Enable new users to register to the system, inserting username, password and email.

[REQ2] Enable a registered user to change his password and his username;

[REQ3] Enable a registered user to ask his password if he forgets the old one.

[DOM0] Username must be unique

3.2.1. [G1] Major functions of the Calendar

[REQ1.0] Allow user to add an appointment or a break with a time-duration into the calendar

[REQ1.1] Inform the user if the programmed appointment is too far to arrive in time;

[REQ1.2] Enable the user to set up different UL for the appointment;

[REQ1.3] Allow user to log into the system and see his own calendar.

[REQ1.4] Allow user to add unreachable events into the system.

[DOM1.0] All events added are added after the current date

[DOM1.1] The system allows the user to add overlapped events. For those, he won't calculate the ETA. Still, a path can be retrieved.

3.2.1.1 [G1.1] Give the possibility to schedule breaks or lunch

[REQ1.5] New breaks or lunch should not overlap into an appointment

[REQ1.6] Allow user to insert a flexible lunch which is a particular break

[DOM1.2] Breaks can be in one place (eg. in a park) or in none.

3.2.1.2 **[G1.2]** *Set level of importance of an appointment*

[REQ1.7] Allow user to receive different notifications based on UL

3.2.2.0 [G2]Major functions of the Travelling System

[REQ2.0] Enable a registered user to set up favorite transportation means or means to activate or deactivate as a general setting (including bike and car sharing);

[REQ2.1] Notify the user that he won't be able to reach an "unreachable event" but still show ETA and path.

[REQ2.2] Enable a registered user that is adding an appointment to choose preferences about the vehicles to use to reach the new appointment.

[DOM2.0] No GPS will be used to determine the current position

[DOM2.1] Because of [DOM2.0], the information given by the travelling systems concerns only pre-trip support in order to accomplish what user asks. Further information on any kind of delays won't be retrieve to the user

[DOM2.2] When using external APIs their information will have an amount of precision sufficient for our needs.

[DOM2.3] ETA will be shown in hours/minutes format.

3.2.2.1 [G2.1]Allow clients to filter solutions

[REQ2.3] Recommend the user a path, based on the ETA, weather, cost;

[DOM2.4] For the moment only few filters will be available

3.2.2.2 [G2.2] Show all available travel solutions to arrive at the next appointment, considering filters.

[REQ2.4] Enable a register user to see different paths to go to the next destination;

[REQ2.5] Enable a user to choose among different solutions where the first ones are displayed considering user preferences

3.2.3.[G3] Major functions of the Ticketing System

[REQ3.0] Enables a registered user to be redirected to buy a ticket online

[REQ3.1] Enables a register user to add a pass to his profile

[DOM3.0] All public transportation will be considered in Lombardia

3.2.3.1.[G3.1] Add a pass for public transportation

[REQ3.2] The pass can have an expiration date, beyond which it will no longer be in effect

[REQ3.3] Allow the pass to be effective in a certain hour range, outside which it will no longer be allowed.

[DOM3.1] System will not check if the user does indeed have the pass. It will assume that the user owns it.

3.2.4.[G4] Support users in their troubles

[REQ4.0] Allows user to take decisions based on weather

[REQ4.1] Allows user to know when there's a strike notifying them

[REQ4.2] Allows user to be notified when there is an accident on the path

[REQ4.3] Allows user to notify the system of an accident

[DOM4.0] Once a strike is declared, it is not revoked. Considering the strike, in general there's a risk of unavailability (not a certainty) of some travel means.

[DOM4.1] To notify the system, the user must have been close to the reported zone

[DOM4.3] There won't be more than 3 events per minute.

[DOM4.4] We assume the existence of a 3rd party API that tells us when there's a strike

3.2.4.1.[G4.1] Tell the user what weather is in a certain day

[REQ4.3] Show users what weather will be in a certain day, in the next two weeks.

[DOM4.2] Weather will be precise

3.3. Performance Requirements

The system needs to be extremely rapid (maximum 10s of milliseconds).
Users not only want to create an event, but usually also get live information about the traffic or when they will arrive in a certain location, therefore they will receive live information on the traffic, coherently with our assumptions.
Users also may rely on the application when they're in difficulty or when they don't want to spend time looking themselves at information.

3.4. Design Constraints

3.4.1. Hardware limitations

Since the application will have to exchange often informations to the server, the user will have to properly set his connection.

3.4.2. Any other constraint

The system is accessible through an app, so in order to be used the user has to have a smartphone with an updated operating system (e.g. Android or IOS).

3.5. Software System Attributes

3.5.1. Reliability

The system *Travlendar+* has to guarantee the persistency of data inserted by the user, like personal data or appointments.
Travlendar+ will guarantee the maximum precise travel time, considering the most number of variables possible. Because of [DOM2.1], some differences between the real time travel and the ETA estimated and the beginning of the travel are still possible. It has to contains the lowest amount of bugs, the MTBF has to be extremely low, and the MTTR has to be relatively fast, of course the graver the error faster must be the correction.

3.5.2. Availability

Travlendar+ has to be available 7 days per week, 24 hours per day considering the constraints mentioned before.
Since the system has to help users during all their travels, at every hour of the day, it will be completely automatic.

3.5.3. Security

The system has to be safe, allowing users to enter a personal space, using their own *id* and *password,* so it has to avoid the possibility of stealing the profile of others users. Furthermore, *Travlendar+* offers the service to buy tickets online, using bancomat or other cards prepared for the online payment, by directly connecting the user to the appropriate external websites. Because of this, the connection must be safe and the user's data protected.

3.5.4. Maintainability

With new technologies invented everyday, it's really hard to estimate an application lifecycle time because it will depend on how market will react.
Travlendar+ has to guarantee an updated system, therefore should be relatively easy to add new functionalities and allow users to use new means of transport.

3.5.5. Portability

Travlendar+ has to be supported by all devices (smartphones, computers, tablets), focussing mostly on smartphone and mobile devices, because they're more comfortable and easier to use while travelling.

3.5.6. Usability

Travlendar+ should be extremely easy to use, the user has only to know the destination and the system will work on the travel.
For more complicated functions, like deciding the favorite paths or the ones to avoid, the user has to try a few times before the app will become easier to understand.

Chapter 4 - Formal Analysis

4.1. Actors

We can define two different actors who interact with the system: the user and the external information.
These two actors have a different role in our system. Here follows a brief description of both of them:

- *User:* he is the one who directly uses the system. He has an account and uses Travlendar+, he adds appointments, buies tickets…

- *External information:* it has a central role in our system during the path creation, it shares APIs with the system, allowing the system to give to the user the best path to meet his needs.

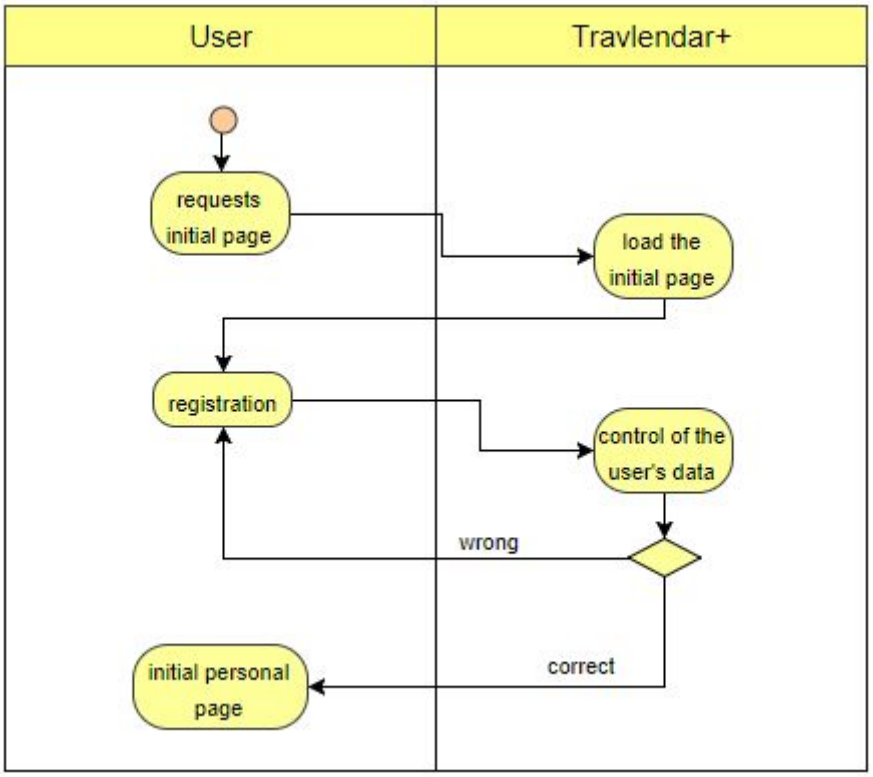
These are only some of the characteristics of the interaction between the system and the actors. Below are described some of the scenario of the system, providing a deeper analysis of the potentiality of our system.

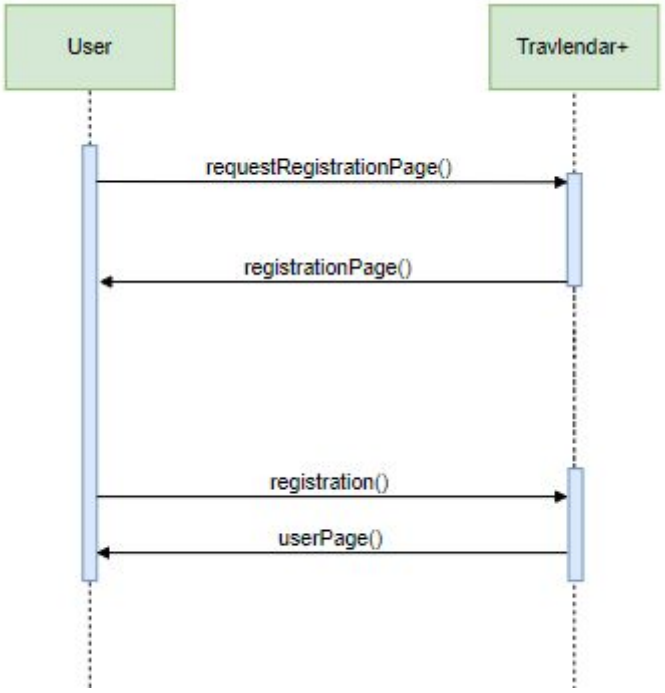
4.2. Scenarios Identification

In this section, some possible scenarios are identified.

4.2.1 Registration and first access

Luke is a veterinarian. He just started his job. For the moment, he's doing a home-service visiting patients directly at their home. Luck hears about Travlendar+ and decides to download the application. First of all he registers and the system checks whether the credentials are already used on another account. If it doesn't find any error, the registration is done successfully.





4.2.2 Break

Mark has just founded his start up and everyday he has a lot of work to do. In this period, his schedule is pretty tight and he has to manage by himself all his meetings and appointments. He can do that quite well, but many times he has to skip the lunch because he realises he doesn't have enough time for it! Therefore he decides to

download Travlendar+ and to register. After the login, he starts to insert all his appointments for the upcoming week. He also decides to reserve some time for himself by adding a "flexible lunch" on each midweek day. He's then asked to add in which time slot he wants to insert the lunch and the amount of time the break should last. The day after, the app will send a notification and tell him the perfect time-period when he can have 30 minutes dedicated to his lunch, without missing any of his appointments.

4.2.3 Weather

Manuel is a sales representative for a big company and he has to travel a lot because of his work. Many times he doesn't know the path he has to take to travel from one appointment to another and how much time it would take to get to the meeting.
Since he started to use Travlendar+, everything is easier. On a rainy day, Mark has to go from one side of the city to the other. He loves driving his motorbike and most of the time according to the app this is the shortest way. However, because of the weather today he doesn't want to take his motorbike and he doesn't know which is the shortest way to arrive at the location of his next appointment. He then consults Travlendar+, which knows about the weather conditions and recommends him to take the subway and the bus to arrive at the destination on time without getting soaked in the rain.

4.2.4 Online Ticket Service

Matt is coming back in Milan after a long time. He has just arrived at the central station but realises it's Sunday and that ticket offices are closed. So he goes to the automatic ticketing service and since "bad luck never travels alone", none of them is working. Because he doesn't want to be late, he's thinking whether it is a good idea to take a taxi. Before taking the decision, Matt consults Travlendar+ to see if there is another option available. He discovers that the metro, as he knew before, is the cheapest way to arrive at his destination on time, but he also finds out he can buy the tickets on Travlendar+ and then decides to do so. After the login, the app shows to the user the best paths to arrive at his destination on time and the associated ticket price (if there is one). Matt decides to choose the fastest path, which requires a metro ticket. After selecting the option of the online buying, he's redirected to the appropriate website to finalize his purchase and can then travel for his destination without the fear of being late!

4.2.5 Adding an appointment

John is a busy person. He has always something to do and somewhere to go and it is not really easy to manage all the appointments by himself. Today he has a meeting that is scheduled between other 2 appointments, and he is not sure if he can manage to be on time at all of them.

John logs in Travlendar+, inserts the new appointment adding day, hour and UL. The system adds the new appointment and notifies John that there are paths to attend all the appointments.
John controls the path and confirms the one he prefers.

4.2.6 Adding a pass

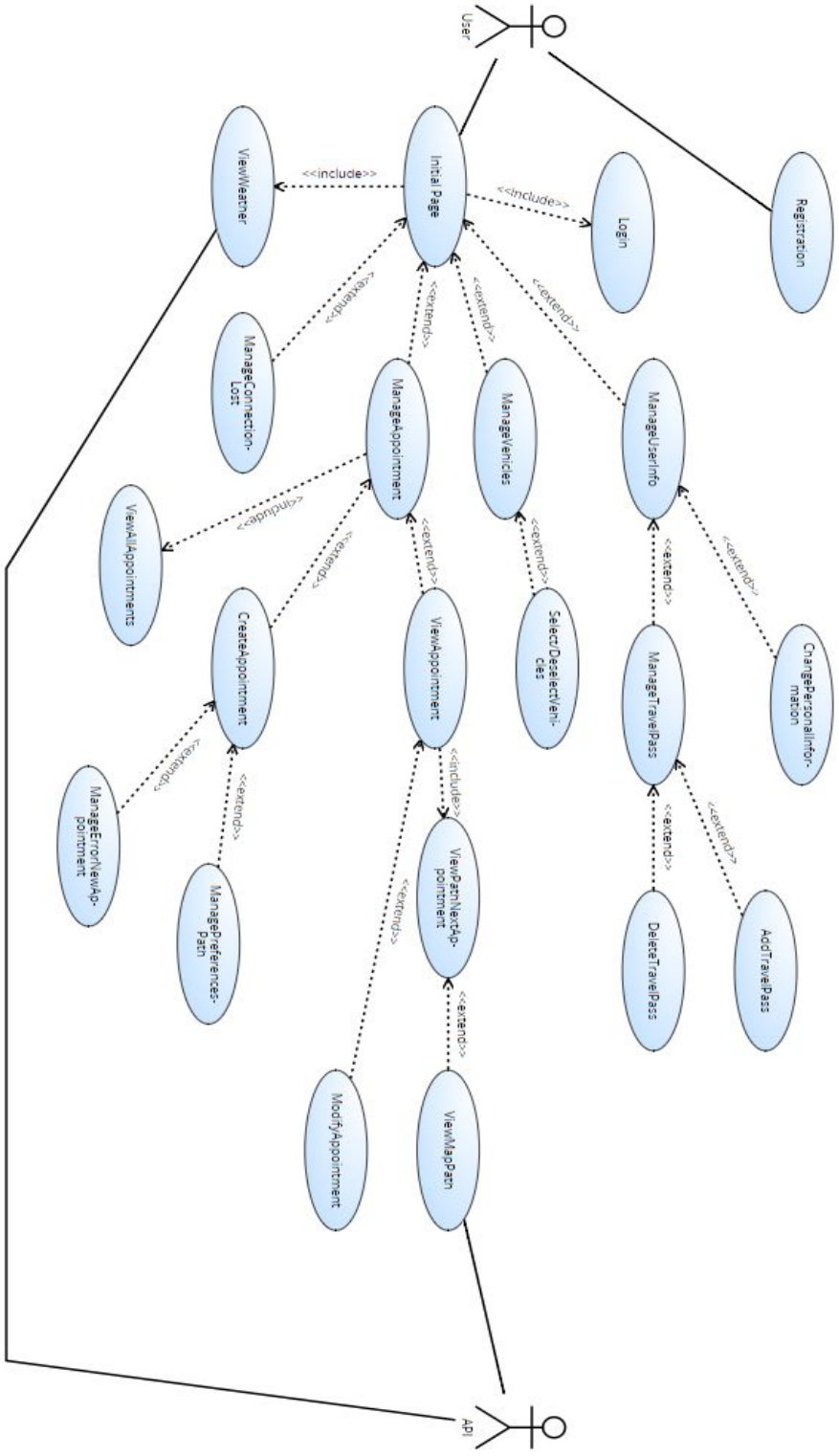
Susanne is a woman who has just moved to Milan and she works in the city center. For all her meetings, which are in different locations, she uses Travlendar+ to manage all her appointments and to always be on time by taking the shortest path.
She doesn't have a drive license, so when she started using the app she decided to customize her profile by globally deactivate the fact that the application has not to suggest her paths which have a travel by car. She also decided to select, as a default option, that the suggestions should try to minimize the carbon footprint and travel cost. Because she doesn't have any pass for public transportations, the app usually suggest her to travel by bike, especially because it is summer and the weather is warm.
However, one day she decides to purchase the annual pass for the public transportations and wants the app to take this into account. She adds the pass to her existing passes, specifying the expiration date and the type of it.
From the next day on, the app can see that Susanne can now take any public transportations without the need of buying a ticket and starts suggesting her taking the subway and the bus to move around the city.

4.2.7 Car sharing

Luke is an offsite student in Milan and even if he has the drive license he has his car in his hometown. He uses Travlendar+ and therefore he has decided to globally deactivate the car as a travel transportation, but he still wants to see travel which contains a shared car. Instead of the car he usually uses his bike, which allows him to travel quickly from one location to the other.
Today he has to go to an important meeting with one of his professor, but when he looks for his bike in the spot where he left it he realises that it was stolen! Worried that he can't make it on time, he checks whether Travlendar+ has any suggestion. The app shows him there is a shared car really near his current location and shows him the path to get to his meeting by car. Relieved, he decides to take the car suggested by Travlendar+ and follows the proposed path, arriving on time to his appointment!

4.3. User's use cases

Below there is a list of relevant use cases for the users. The actor in this use cases is always the user.


User
Initial Page
ViewWeather
Login
Registration
ManageConnection-Lost
ManageAppointment
ManageVehicles
ManageUserInfo
ViewAllAppointments
CreateAppointment
ViewAppointment
SelectDeselectVehi-cles
ChangePersonalInfor-mation
ManageTravelPass
ManageErrorViewAp-pointment
ManagePreferences-Path
ViewPathNextAp-pointment
DeleteTravelPass
AddTravelPass
ModifyAppointment
ViewMapPath
API
<<include>>
<<include>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>

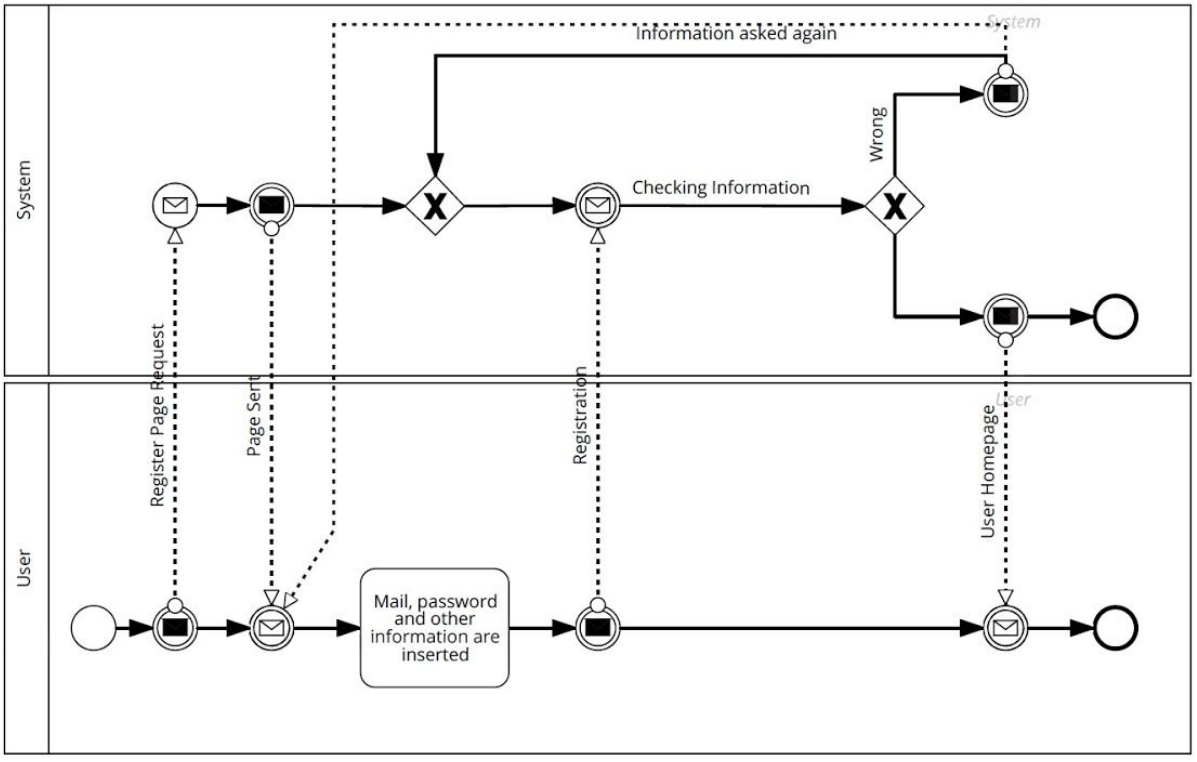
4.3.1 Use Case: "User registration"

Precondition: The user has already downloaded the application, or she/he is connected to the web portal.

Flow of events:

- The user provides a mail and password. [*Figure 3.1.1.1]*
- The user gives information about himself (name, surname, date of birth, home address, gender, username and password)
- The user clicks the button to sign up, and the registration is submitted to the system.
- The system checks whether the username is already used.
- The system verifies if the email is valid or if it is already used.
- The system checks if the password is strong enough.
- The systems tells the user that the registration was successful and sends a "welcome e-mail" where it confirms the registration.
- The system logs the user in.

Exit condition: the user is registered and logged into the system.
Exception: if the user submitted data that are invalid, or the username is already used the system will ask them again.



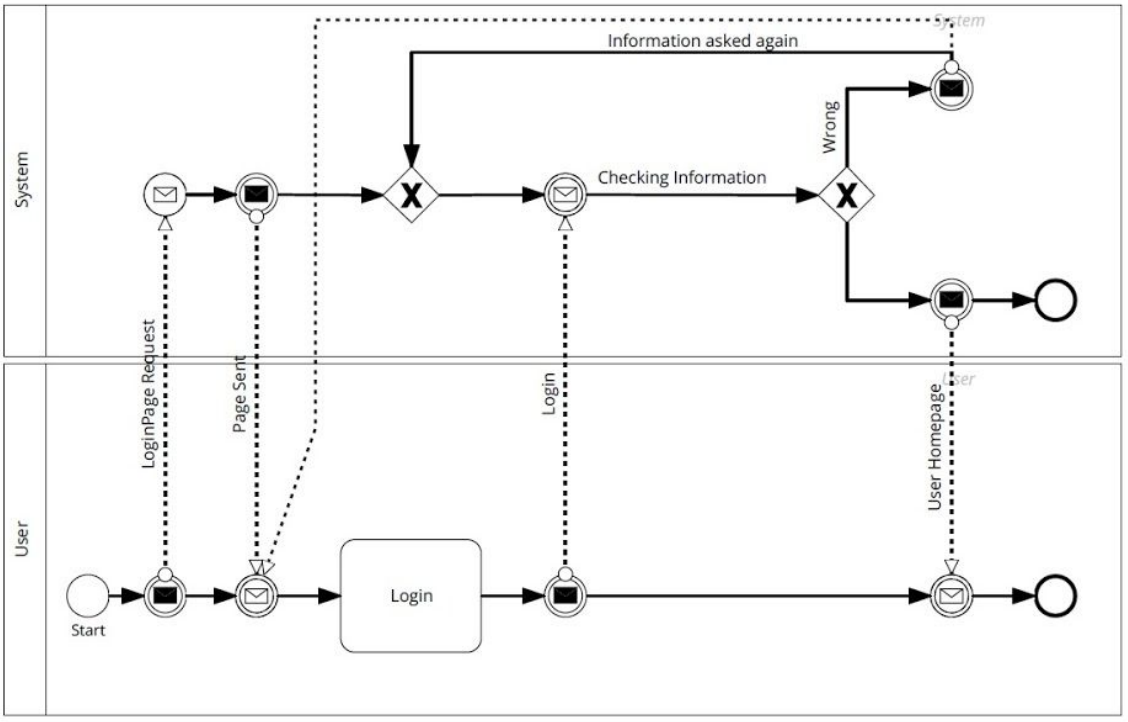
4.3.2 Use Case: "User login"

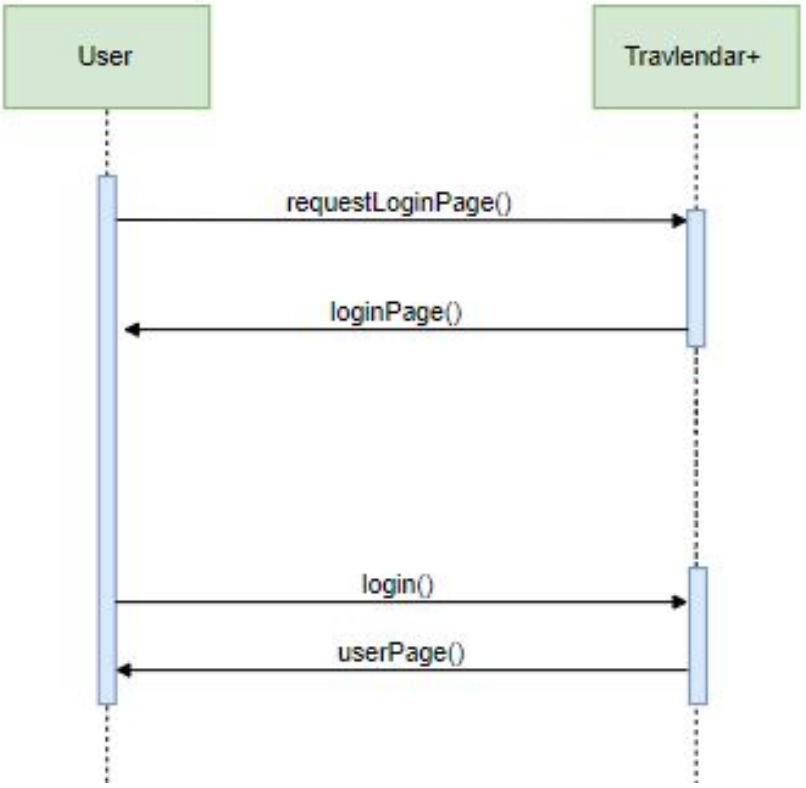
Precondition: The user has downloaded the application, or she/he is connected to the web portal. The user is also registered.

Flow of events:

- The user insert username and password. [*Figure 3.1.1.2]*
- The user clicks the Login button.
- The system checks the credentials.

Exit condition: the user is logged into the system and the initial page is shown to him.
Exception: if the user inserted invalid credentials, the system will ask again for them.





4.3.3 Use Case: "Adds an event"

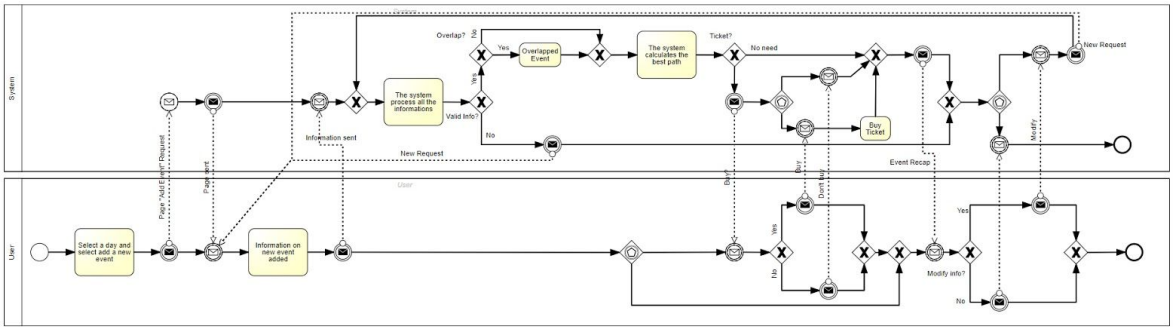
Precondition: The user has already logged in.

Flow of events:

- User selects a day. [*Figure 3.1.1.3 - 3.1.1.4]*
- The system shows all the events in that particular day. [*Figure 3.1.1.9- 3.1.1.6]*
- The user clicks "add an event" button.
- The system displays the "add event" window. [*Figure 3.1.1.7]*
- The user fills all the information needed.
- The system checks whether all information are semantically corrected. If no "change start" information is provided by the user, the system will use the information of user's last appointment.
- The system then calculates the user's best path with the information the user has given and automatically chooses the vehicles. The system displays a recap of the event.
- The user sees the best path proposed by the application.
- If the user wants to use a public vehicle, the system will check if the user has already a pass. If user hasn't, the system will show a warning message telling the user that he needs to buy a ticket for the travel. In the message, there are also 2 buttons where the user can decide to buy the ticket himself.

Exit condition: User added a full event to the calendar
Exceptions:

- If there is an overlap of events, the system will notify the user who will still be able to continue the “add event” operation and add an “overlapped event”.



4.3.4 Use Case: “Edit an event”

Precondition: The user is logged in and has an event in his calendar he has not already attended.

Flow of events:

- The user selects a day with at least an event from the calendar.
- The system shows all the events in that day.
- The user double clicks (PC) or taps (Mobile) into an event name.
- The system shows the details of that event.
- The user modifies the information of the event.
- The system checks if all information are sintattically corrected and acts as if a new event has been submitted doing similar operations to [4.3.3]

Exit condition: User added modifies an event from the calendar
Exceptions: A user can’t try to modify an event that has already passed. All the exceptions here come from [4.3.3].

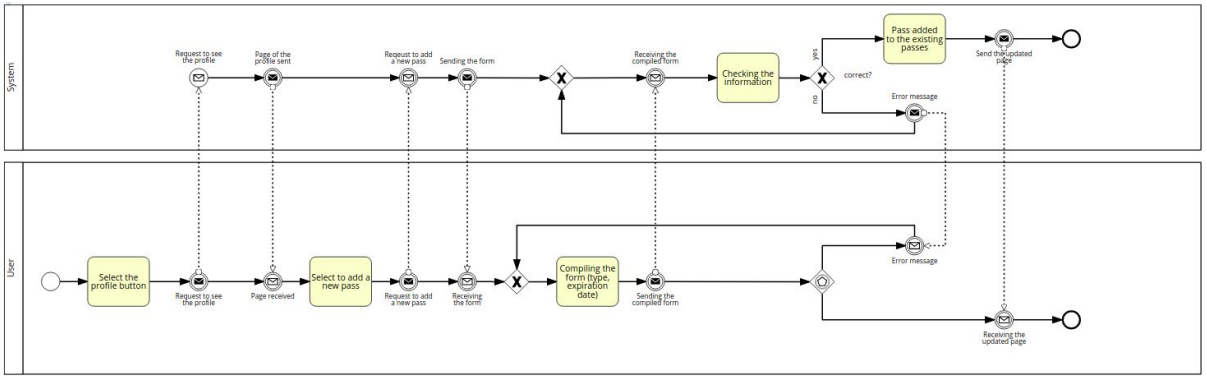
4.3.5 Use Case: "User adds a pass to his profile"

Precondition: The user is logged in.

Flow of events:

- The user selects the profile button.
- The system displays the "profile window".
- The user clicks the "Add pass…" button.
- The system display a window where it asks what type of pass the user has.
- The user fills all the information.
- The system checks if the information are semantically correct and accepts them.

Exit condition: User added a pass to his profile.
Exceptions: If provided information isn't correct, the system asks the user to insert them again.



4.3.6 Use Case: "Insert a break event"

Precondition: The user is logged and the "day window" [*Figure 3.1.1.6 - 3.1.1.9]* is shown to him.

Flow of events:

- The user selects "add an event" button.
- The system displays the add event window.
- The user fills all the information needed and selects the "break" button. In particular, he sets the interval when he wants to have a break ,the day(days) he wants to do it, the location (optional) and how long wants it to last.

- The system checks if all information are semantically corrected. If no “start location” [*Figure 3.1.1.7]* information is delivered, he will use the information of user’s last appointment.
- The system then computes the information given by the user and sets up a break.

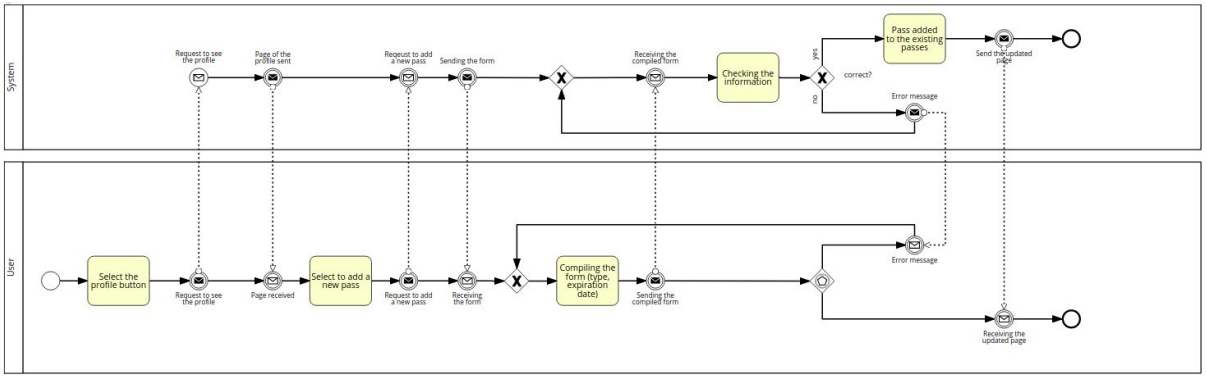
Exit condition: User added a break into the calendar
Exceptions: If provided information isn’t correct, the system asks the user to insert them again. If there’s no room for a break, the system will notify the user.

4.3.7 Use Case: “View the travel path”

Precondition: The user has just logged in and has at least two appointment in the same day. [*Figure 3.1.1.9]*

Flow of events:
- The user selects a particular day where he has at least two appointments.
- The system displays the day schedule.
- The user sees a brief recap of the travel from one appointment to the other, concerning the vehicles he will have to use and when he will have to depart if he wants to arrive on time. [*Figure 3.1.1.10]*
- The user then clicks the recap.
- The system displays a new window with all the information of the path.

Exit condition: All the information about the path are shown to the user.
Exceptions: No exceptions occurred.



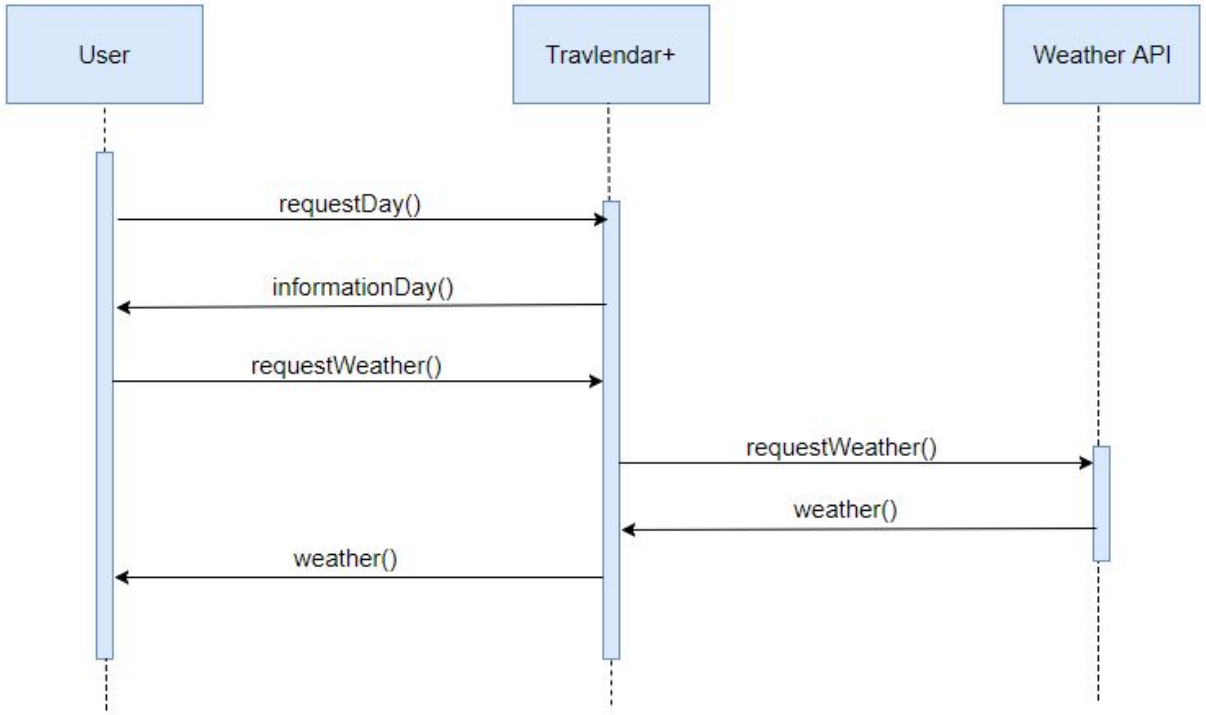
4.3.8 Use Case: "View weather"

Precondition: The user is logged in and the calendar is shown to him.

Flow of events:

- The user is on the "calendar window". Here, there's an icon showing what the weather will mainly be in that day. The user selects a certain day.
- The system displays day window.
- The user clicks the weather icon.
- The system displays the daily weather icon.

Exit condition: User saw correctly what will be the weather in a particular day, based on the weather API[1.5.3].
Exceptions: No exception occurred.



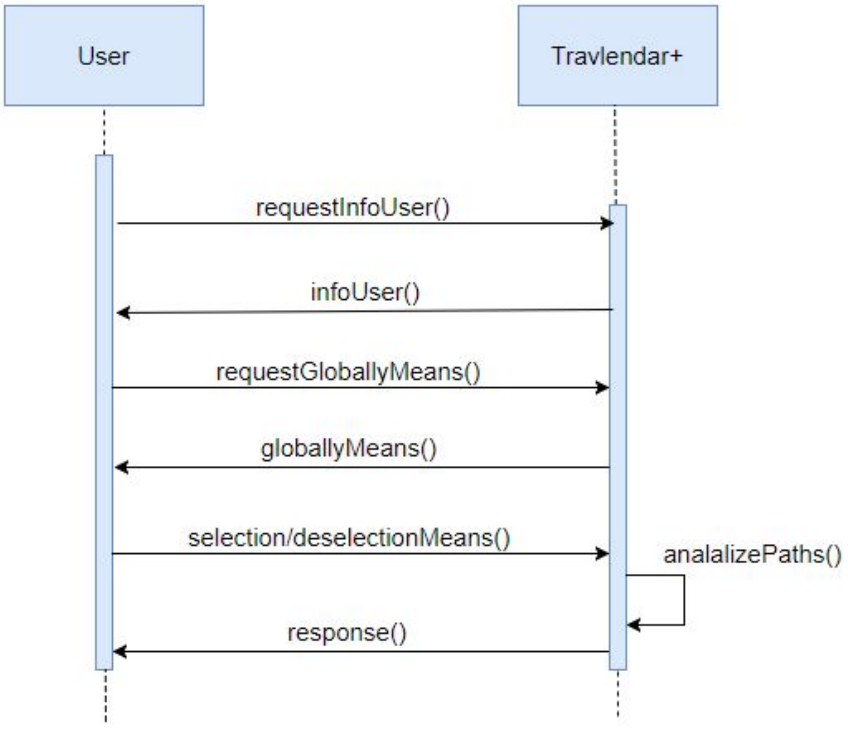
4.3.9 Use Case: "Edit globally means"

Precondition: The user is logged in.

Flow of events:

- The user selects the profile button.
- The system displays his profile settings.
- The user selects his vehicles preferences.
- The system displays all the transportation means he can active or deactivate.
- The user selects the means to activate and/or deactivate.

Exit condition: The system saves the means of transport the user has deactivated.
Exceptions: If the user has an appointment in his calendar which uses some means of transport that have just been deactivated, the system notifies the user.
The user then can cancel the operation, or let the system modify the travel paths, according to the last changes. The user can still modify them manually [4.3.4].



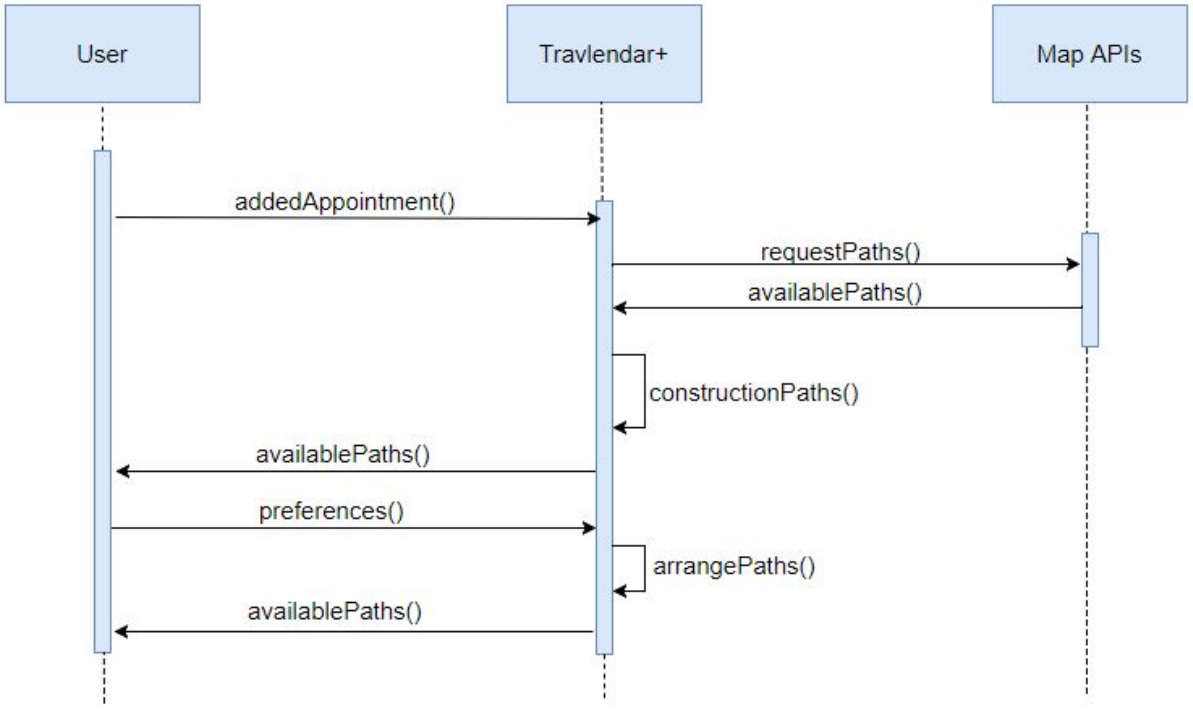
4.3.10 Use Case: "Select preferences on the vehicles"

Precondition: The user is adding an event.

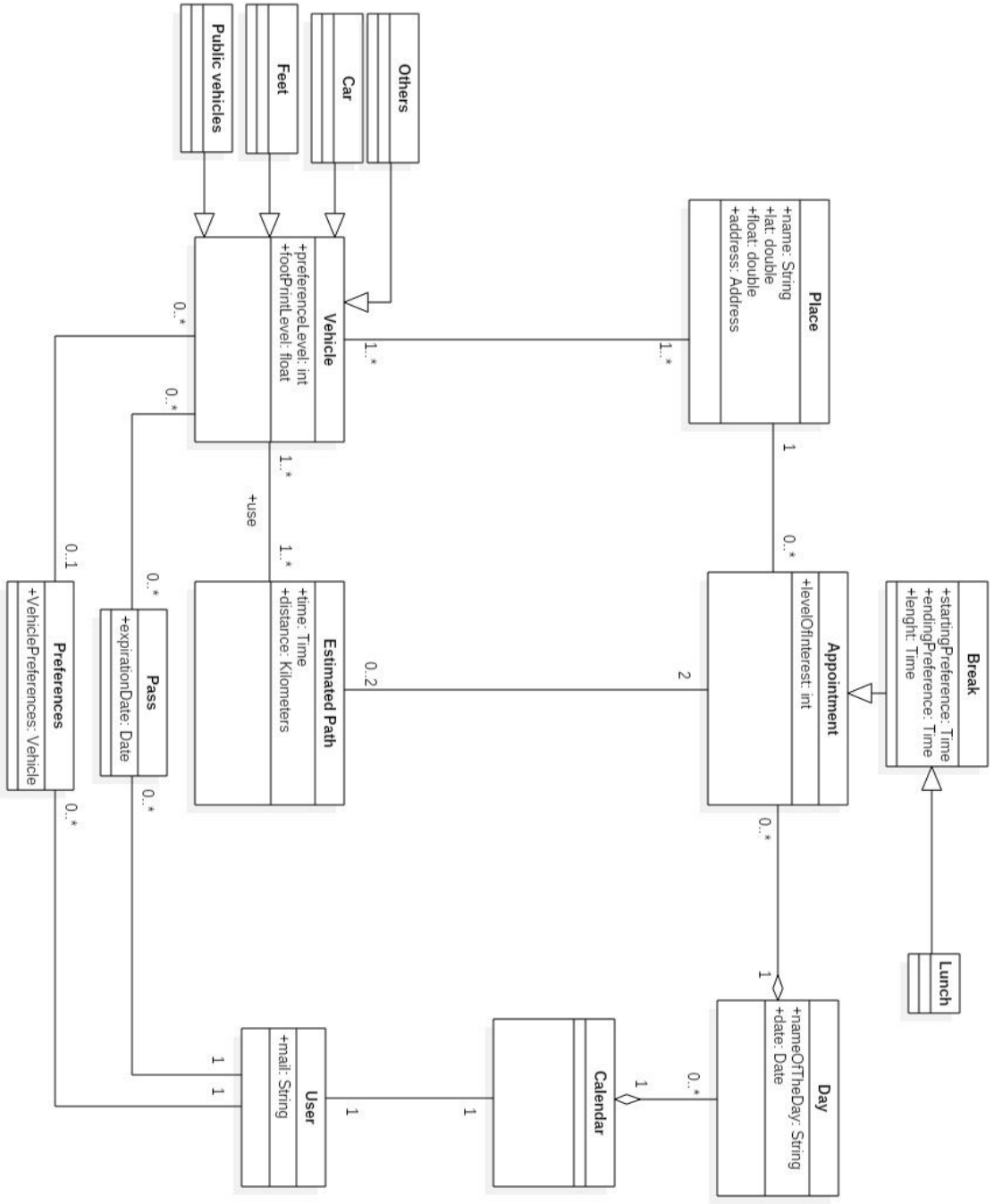
Flow of events:

- The user sees the appointment he has just added.
- The system recommends to the user the best path to reach the appointment
- The user can add certain preferences for this particular path
- The system analyze the preferences and updates the recommended path

Exit condition: User saw the different paths recommended by the system, among them the paths in evidence will be the ones that contains the user's preferences



4.4. Class diagram



4.5. Alloy

4.5.1. Model code

```
module Travlendar

sig Date{}

sig Code{}

sig Hour{
        hour: one Int,
        minute:one Int}
        {
        hour>=0
        hour<=7
        minute>=0
        minute<=7}


sig Coordinate{}

sig User{
        username:one Code,
        password:one Code,
        pass:set PassVehicles}

sig PassVehicles{
        type:one Category,
        startingHour:lone Hour,
        endingHour:lone Hour,
        expirationDate:lone Date}

sig Day{
        data: one Date,
        appointments:set Appointment}

sig Calendar{
```

```
        personalUser:one User,
        day: set Day}

sig Place{
        coordinate:one Coordinate
}

sig Break extends Appointment{
        until:one Hour}

sig Vehicle{
        name:one String,
        start:one Place,
        end:one Place
}{start!=end}

sig Appointment{
        place:one Place,
        hour:one Hour,
        ul:one Int
}{ul >= 1 and ul <=5}

sig Path{
        startPoint:one Appointment,
        endPoint:one Appointment,
        means:set Vehicle,
        duration:one Hour
}{
        startPoint != endPoint
        #means >= 1
        all x:means | one x1:mans | x.end = x1.start or

        x.end = endPoint.place
        all x:means | one x1:means | x.start = x1.end or

        x.start = startPoint.place
}
```

In this part of the alloy code we define the main signatures of the system.
The constraints of the definition of the signatures are already specified in this part of the code.
These are the signatures of the system:

- Date, Code and Coordinate:
  are generic signatures, used to define following signatures
- Hour:
  indicates the time at which the appointment is scheduled
  - hour:
    the hour of the scheduled appointment
  - minute:
    the minute of the scheduled appointment
- User:
  the person who is registered in the system and uses the functionality
  - username:
    name of the user to be recognized by system
  - password:
    secret string to verify the username
  - pass:
    the pass for the vehicles to use freely
- PassVehicles:
- the pass to use a vehicle freely
  - type:
    category of vehicle to use
  - startingHour:
    since when the vehicle can be used
  - endingHourd:
    until when the vehicle can be used
  - expirationDate:
    when the pass will lose validity
- Day:
  one of the day presents on the calendar of the user
  - date:
    the day, the month and the year of this day
  - appointment:
    is the relation that links this day with the scheduled appointments
- Calendar:
- the calendar of the user, the main functionality of the system
  - personalUser:
    connects the calendar with the user registered to the system, every calendar has an single user
  - day:
    all the days that are in the calendar
- Place:
  a place in the physical world
  - coordinate:

the coordinates of the place

- Break:
  a particular type of appointment
  - until:
    specify how far the user is willing to wait
- Vehicle:
  is the vehicle recommended to the user to travel from a place to another one
  - name:
    the name of the vehicle
  - start:
    the place where the user starts to use the vehicle
  - end:
    the place where the user ends to use the vehicle
- Appointment:
  is the user's scheduled meeting, saved in the system
  - place:
    where the appointment takes place
  - hour:
    when the appointment takes place
  - ul:
    the degree of importance of the appointment
- Path:
  the route the user is recommended to take to go to the next appointment
  - startPoint:
    the appointment that the user has just finished
  - endPoint:
    the next scheduled appointment
  - means:
    the vehicles the user has to take to reach the next appointment
  - duration:
    the time needed to go to the next appointment

4.5.2. Constraints Code

This fact means that every user in the system has to use different username:

```
//the username is unique
fact unrepeatable{
        all disj u1,u2: User | u1.username != u2.username
}
```

Cannot exist calendars used by the same user, if a physical user wants to have 2 or more calendar he has to register with different username:

```
//the calendar is unique for every user
fact userHasAUniqueCalendar{
	all disj c1,c2: Calendar | c1.personalUser != c2.personalUser
}
```

The system gives to every user a calendar to insert the appointments, it does not exist a user not linked to a calendar:

```
//every user has a calendar
fact userHasAtLeastACalendar{
	all u:User | one c:Calendar | u = c.personalUser
}
```

In the calendar all the days are different, does not exist 2 days in the same calendar that has the same date:

```
//the calendar has not day with the same date
fact uniqueDayInCalendar{
	all disj d1,d2:Day | all c: Calendar | (d1 in c.day and d2 in c.day) implies d1.date
!= d2.date
}
```

This is a constraint due to the physical world, a person cannot be in two places at the same time, so the appointments are defined so that cannot exists a user that has 2 appointment at the same instant:

```
//cannot be 2 appointments at the same moment
fact distinctAppointments{
	all d:Day | all disj a1,a2:Appointment | (a1 in d.appointments and a2 in
d.appointments) implies a1.hour != a2.hour
}
```

The calendar is complete of all the days:

```
//the calendar has all the day
fact calendarIsComplete{
	all dat:Date | all c:Calendar | one d:Day | d in c.day and d.date = dat
}
```

There are also different constraints defined during the constructions of the signature:

- {
  start!=end
  }
  the vehicles of the system cannot have the same place of departure and arrival, the vehicle is used to describe how is defined a path between two places, if the starting place is the same of the ending place a vehicle is not needed
- {
  hour>=**0**
  hour<**7**
  minute>=**0**
  minute<**7**
  }
  this constraint describes the system time of our system which obviously has minutes and hours that must meet certain constraints,
  obviously minutes should have as a maximum limit of 60 and hours should have 24, but because of alloy it has to be set at 7
- {
  ul >= **1** and ul <=**5**
  }
  this constraint describes the urgency levels of our system when the user insert an appointment
- {
  startPoint != endPoint
  #means >= 1
  all x:means | **one** x1:means | x.end = x1.start or

  x.end = endPoint.place
  all x:means | **one** x1:means | x.start = x1.end or
  x.start = startPoint.place
  }
  these constraints are used to define the path, the path is an union of more vehicles, because between two appointment the path that the system advices to the user is the fastest, and is possible that the user has to change different vehicles during the path, the path has a starting place and a different place of arrival, the user has to take at least a vehicle(walking is considered a vehicle), all the vehicles in the path are connected(if a vehicle ends its race the user will have another vehicle to take or he is arrived at the appointment)

4.5.3. Assertions Code

Defined our constraints, we go to the assertion stage to verify that our system is consistent with the constraints defined in the previous paragraph.

```
//the username is unique
assert unrepeatable{
        all disj u1,u2: User | u1.username != u2.username
}

check unrepeatable for 5

//the calendar is unique for every user
assert userHasAUniqueCalendar{
        all disj c1,c2: Calendar | c1.personalUser != c2.personalUser
}

check userHasAUniqueCalendar for 5

//every user has a calendar
assert userHasAtLeastACalendar{
        all u:User | one c:Calendar | u = c.personalUser
}

check userHasAtLeastACalendar for 5

//the calendar has not day with the same date
assert uniqueDayInCalendar{
        all disj d1,d2:Day | all c: Calendar | (d1 in c.day and d2 in c.day) implies d1.date !=
d2.date
}

check uniqueDayInCalendar for 5

//cannot be 2 appointments at the same moment
assert distinctAppointments{
        all d:Day | all disj a1,a2:Appointment | (a1 in d.appointments and a2 in
d.appointments) implies a1.hour != a2.hour
}
```

```
check distinctAppointments for 5

//the calendar has all the day
assert calendarIsComplete{
        all dat:Date | all c:Calendar | one d:Day | d in c.day and d.date = dat
}

check calendarIsComplete for 5

//the hour is correct
assert correctHour{
        all h:Hour | h.hour>=0 and h.hour<=7 and h.minute >=0 and h.minute<=7
}

check correctHour for 5

//the urgency level is correct in all the appointments
assert correctUL{
        all a:Appointment | a.ul >=1 and a.ul<=5
}

check correctUL for 5

//the paths don't have disconnection
assert noDisconnectedPaths{
        all p:Path |      p.startPoint != p.endPoint and
        #p.means >= 1 and
        (all x:p.means | one x1:p.means | x.end = x1.start or

        x.end = p.endPoint.place)
        and
        (all x:p.means | one x1:p.means |x.start = x1.end or

        x.start = p.startPoint.place)
}

check noDisconnectedPaths for 5
```

All these assert tests in the Alloy Analizer 4.2 software do not generate counterexample, then we can guess that the pattern is correct, even if we do not have the certainty as you try it on a finite set of cases and not infinite.

4.5.4. Functions Code

of the features that our system uses, we have decided to put below only the most important ones:

- the code below describes the addition of a new appointment on a given day, the added appointment is already complete with all the data when given as an input

```
//add appointment in a particular day
pred addAppointment [d:Day, a:Appointment] {
d.appointments = d.appointments + a}
```

- the code here describes adding a user pass

```
//a user add a pass
pred addPass[p:PassVehicles, u:User]{
u.pass = u.pass + p}
```

- the code below describes the action to view all appointments on a given day, the input of the code is the day selected and the output are all the appointments of that given day

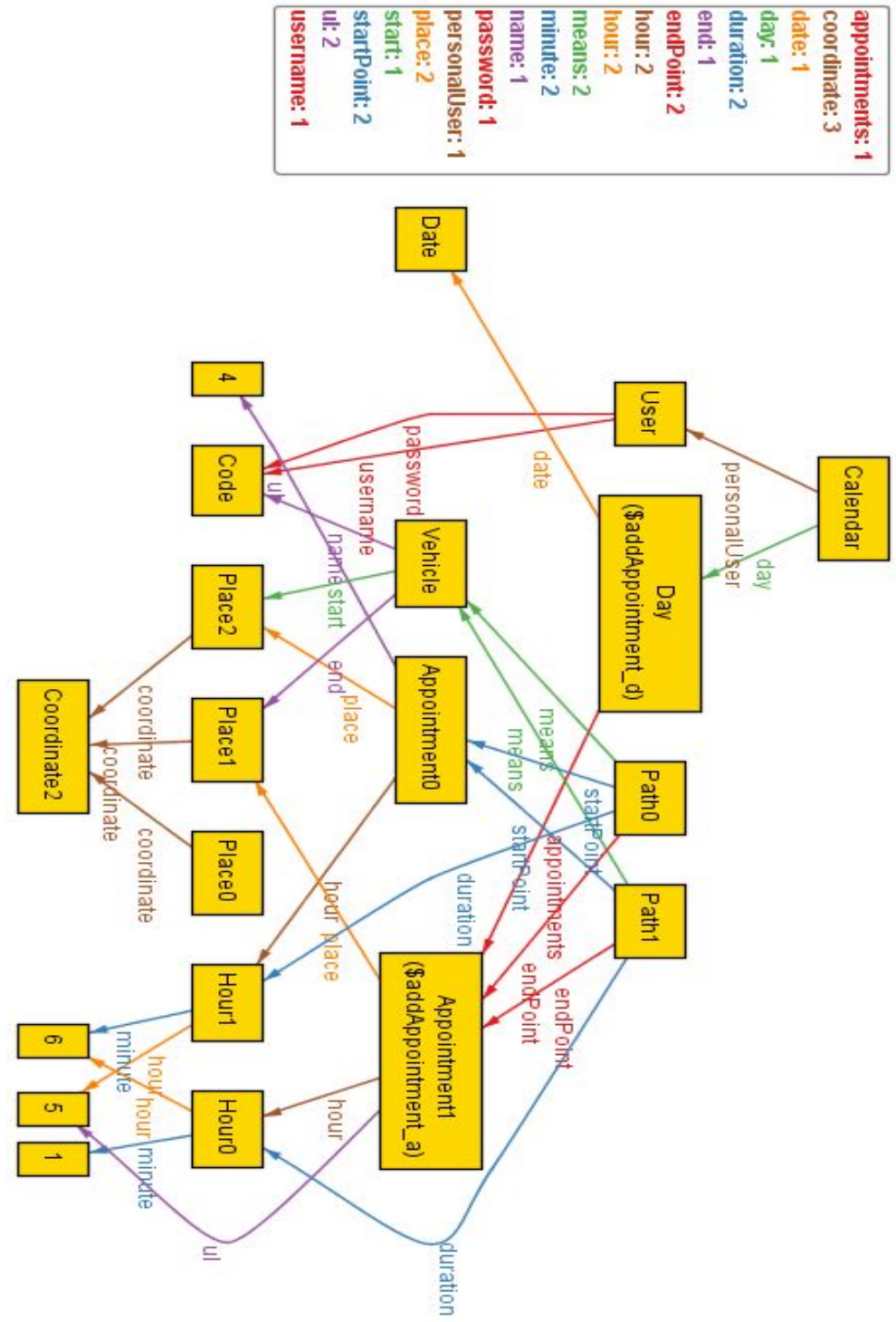
```
//getting all the appointments of a particular day
fun getAppointment[d:Day]: set Appointment{
d.appointments}
```

- the code below serves to create a world that has the following characteristics

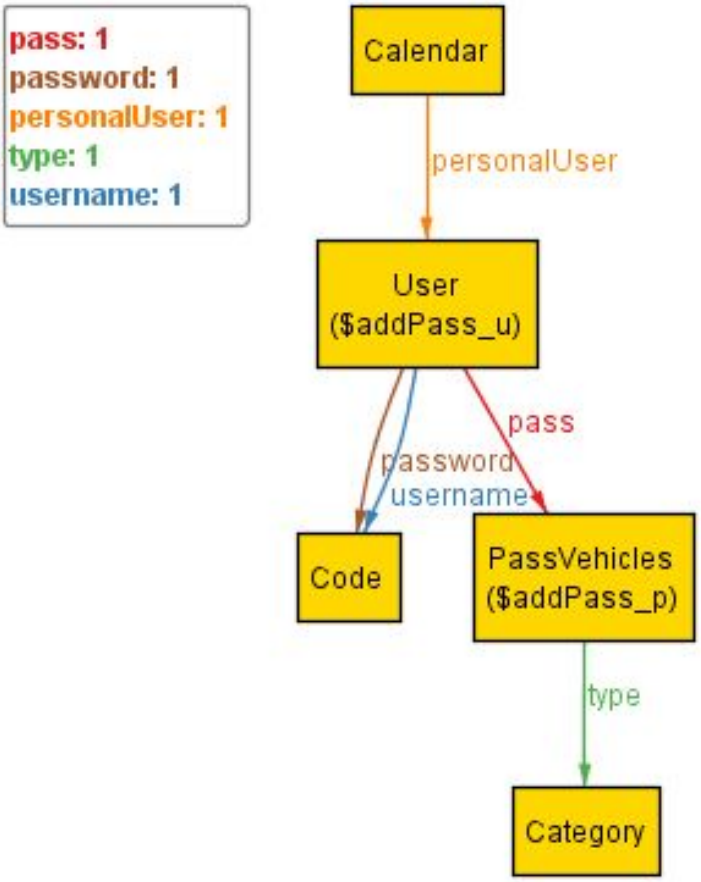
```
pred show{
        #User=1
        #Calendar=1
        #Place>0
        #Appointment>0
        #PassVehicles=1
        #Day>0
        #Break=1
        #Path=1
        #Vehicle=1
}
```

4.5.5. Models of the world

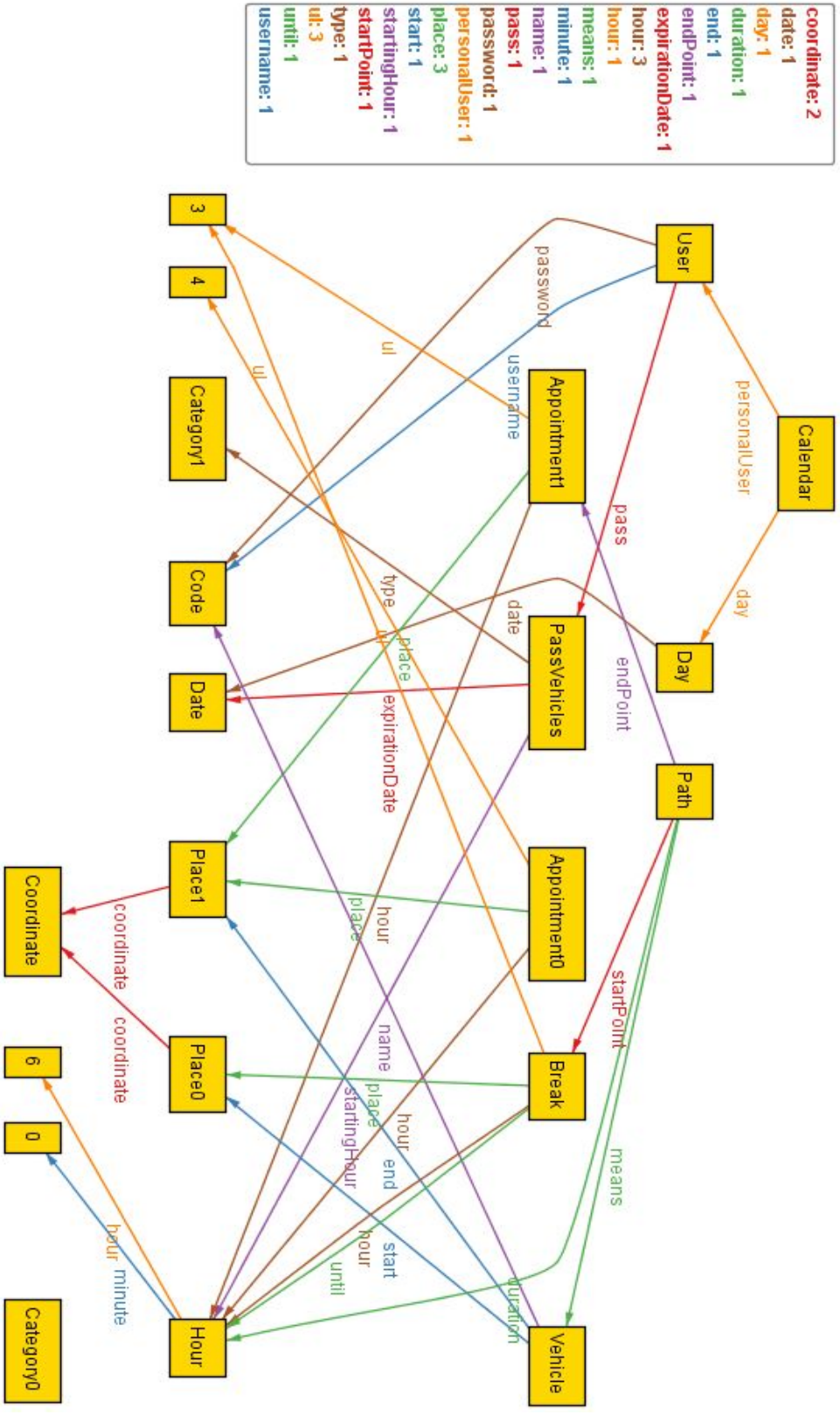
Add of an appointment



Add a Pass


pass: 1
password: 1
personalUser: 1
type: 1
username: 1
Calendar
personalUser
User
($addPass_u)
pass
password
username
Code
PassVehicles
($addPass_p)
type
Category

The world of our system



5. Effort Spent

Students	Hours spent
Federico Amadelli	28.5 hrs
Alessandro Artoni	30 hrs
Alessio Baccelli	22 hrs

6. References

We decided to make a survey, in order to understand the users' preferences (e.g. web application vs mobile application).
We attached this survey to the RASD in the delivery folder.

Samples from last years in beep were used as reference for this work.

Alloy the links that were in the slides, and the samples presented in the lessons, were also used as reference.

7. Tools Used

This document has been realized by using various software tools, of which:

- **Adobe XD CC - Adobe Photoshop CC 2017**
  to create the mockup of the user interface.
- **Alloy Analiyzer 4.2**
  to write and check the alloy code.
- **Draw.io**
  to create sequence diagram and the domain graph.
- **Google documents**
  to write the document.
- **Google surveys-**
  to make the survey.
- **Signavio**
  to create BPMN and use cases.
- **StarUML**
  to create the class diagram.