

Software Engineering 2

Travlendar DD

DESIGN DOCUMENT

Version 1.0 - 27/10/2017

Plamen Pasliev – 898793

Victor Álvaro Gonzalez - 897700

Table of contents

[1. Introduction 4](#_Toc497319637)

[1.1 Objectives 4](#_Toc497319638)

[1.2 Scope 4](#_Toc497319639)

[1.3 Definitions, acronyms and abbreviations 4](#_Toc497319640)

[1.3.1 Definitions 4](#_Toc497319641)

[1.3.2 Acronyms 5](#_Toc497319642)

[1.3.3 Abbreviations 5](#_Toc497319643)

[1.4 Reference documents 5](#_Toc497319644)

[1.5 Document Structure 5](#_Toc497319645)

[2. Overall description 7](#_Toc497319646)

[2.1 Product perspective 7](#_Toc497319647)

[2.2 Product functions 7](#_Toc497319648)

[2.3 User characteristics 7](#_Toc497319649)

[2.4 Constraints 7](#_Toc497319650)

[2.4.1. Regulatory policies 8](#_Toc497319651)

[2.4.2 Hardware constraints 8](#_Toc497319652)

[2.5 Assumptions and dependencies 8](#_Toc497319653)

[3. Specific requirements 9](#_Toc497319654)

[3.1 External interface requirements 9](#_Toc497319655)

[User Interfaces 9](#_Toc497319656)

[3.2 Functional requirements 10](#_Toc497319657)

[3.3 Performance requirements 12](#_Toc497319658)

[3.4 Design Constraints 12](#_Toc497319659)

[3.5 Software System Attributes 13](#_Toc497319660)

[Reliability 13](#_Toc497319661)

[Availability 13](#_Toc497319662)

[Security 13](#_Toc497319663)

[Maintainability 13](#_Toc497319664)

[Portability 13](#_Toc497319665)

[4. Use cases 14](#_Toc497319666)

[5. UML diagrams 22](#_Toc497319667)

[6. Formal analysis using Alloy 27](#_Toc497319668)

[6.1 Purpose 27](#_Toc497319669)

[6.2 Code 27](#_Toc497319670)

[6.3 Results 34](#_Toc497319671)

[6.3.1 Proof of consistency 34](#_Toc497319672)

[6.3.2 Generated world 35](#_Toc497319673)

[7. Effort spent 36](#_Toc497319674)

[8. References 37](#_Toc497319675)

# 1. Introduction

## 1.1 Objectives

This document represents the Design Document (DD). The main goals of this document is to give more details about the development of the application and its design. This way it will be specify the architectural design, algorithm design and user interface design. In addition to this, the order in which the application will be implemented and the tests will be identified. This document is addressed to the developers who have to implement the requirements.

## 1.2 Scope

In this project we are going to develop and implement an application called Travlendar. It is a calendar-based application which allows you to create a calendar according to the events you have such as appointments related to work or personal reasons. On top of calculating the time that the user has between meetings so that he does not arrive late, the application will suggest the best mobility option between events and will alert him when it is impossible to reach a specific event on time. In addition to these functionalities, the system will allow the user to buy public transportation tickets and locating the nearest point to hire another type of service (bike of a bike sharing system, car of a car-sharing service, etc.). Users could define their transportation preferences, they can activate or deactivate any kind of transportation (including walking). The application will also take into account the weather in the location of the user. If it is raining at the time the user has to move to another event, the system will take this into account and will change the way of transport if it is necessary. The application will also allow the user to define breaks to eat or to develop other types of activities. In this way the system will organize the meetings of the user according to their breaks and the time they need to do these activities. Finally, users should also be able, if they wish to, to select combinations of transportation means that minimize carbon footprint.

## 1.3 Definitions, acronyms and abbreviations

### 1.3.1 Definitions

* *Meeting:* personal or work-related appointment. In some parts of the text, meetings are referred as appointments.
* *User*: a user of the Travlendar system
* *Calendar:* a timetable containing meetings sorted by date
* *Alert:* a notification or a pop-up screen.
* *Customer:* user that downloads the application.

### 1.3.2 Acronyms

* *RASD*: Requirement Analysis and Specification Document.
* *API*: Application Programming Interface

### 1.3.3 Abbreviations

* [Gn]: n-goal.
* [Dn]: n-domain assumption.
* [Rn]: n-functional requirement.
* [Pn]: n-performance requirement.

## 1.4 Reference documents

## 1.5 Document Structure

1. In the first part of the document specify what the design document is about. In the same way is explained, without going into much detail, how the application works. Finally, is given some information about definitions and abbreviations to better understand the rest of the document.
2. In the second part it is given an overall description of the system including the functions of the application, clarifying some concepts of the system. Also list the actors who are going to take part of the system. In the same way the constraints and limitations of the system will be defined. Finally, are specified text and domain assumptions to resolve certain types of doubts that may arise by reading the document.
3. The third part of the document refers to specific requirements. We have defined both functional and non-functional requirements. In this part of the document we will go into more detail in the aspects mentioned in section 2.

# 2. ARCHITECTURAL DESIGN

## 2.1 OVERVIEW

In our system we only have one element, the user application that will serve as an intermediary between the user and Google servers to perform certain queries. Thus the user will consult the application and it will communicate it to the servers that will return an information that the system must process and show the user through the graphic interface of the application:

1. User application: represents the user who has downloaded the Travlendar application. The application allows the user to create their own calendar in such a way that they can create, edit and delete events. Within the events, the user will have to specify when creating a new one if this is a break or a meeting. Therefore, the only actors of the application will be the user and the software of the application. The user makes queries to the application through a graphical interface and the software responds with notifications, creating, deleting or editing new events, calculating how to reach the destination taking into account parameters acquired by the user's location (bad weather, strikes, distances, etc.) and user preferences. The application does not have a server or a database because the information is stored directly in the Google account of the user. For this reason, the user must have a google account which must be logged in, in such a way that the Google Calendar API can be accessed through the Google servers. This way we can avoid the purchase and maintenance of a server using this way a safe and cheap data storage such as Google. Saving the information in the user's account so that if the user changes the device they can continue using their calendar and do not have to create the events again.

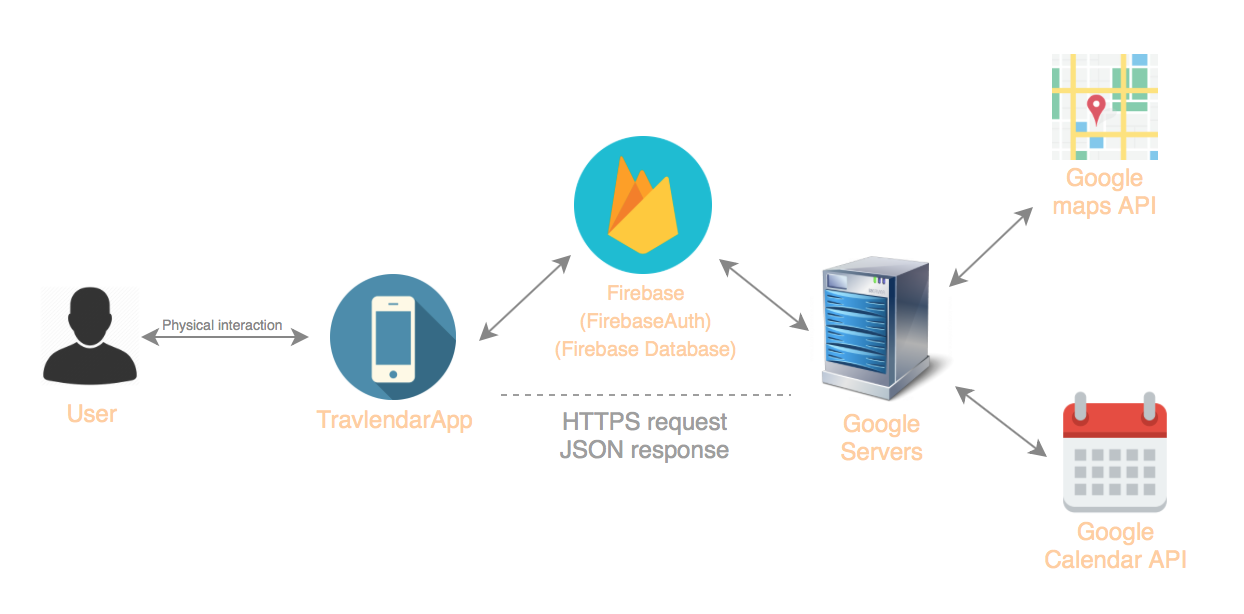
In addition, in the diagram there are also represented the external actors involved in the process of providing the service:

- Google Maps: representation of the Google Maps Web Services. In particular:

* GeoCoding API for the conversion of a string of text into a GPS position.
* GeoLocation API for the GPS localization of devices.

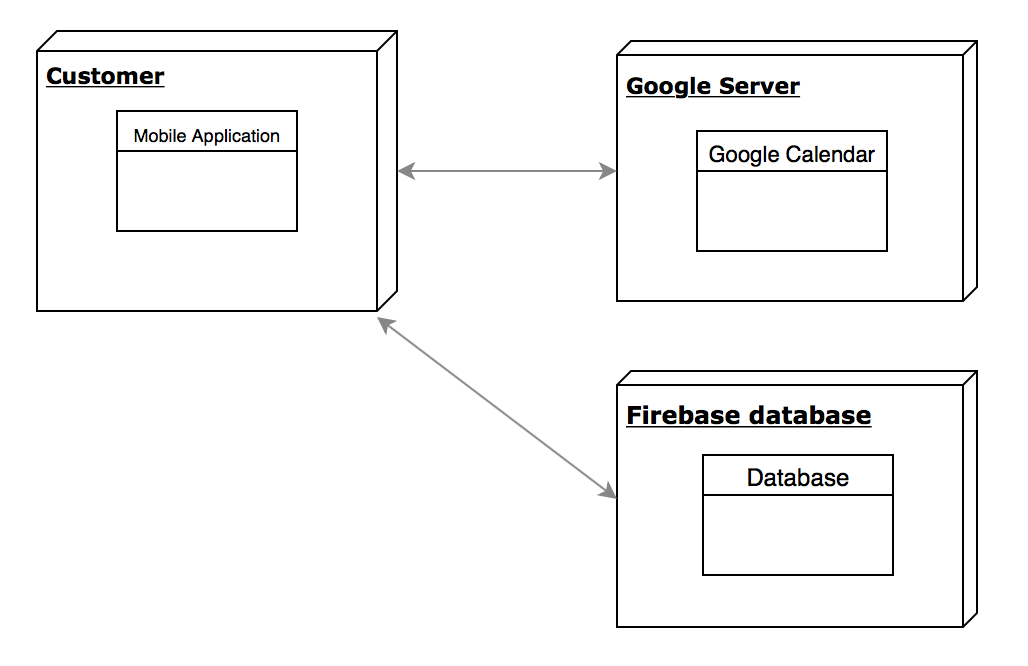
- Google Calendar API: manipulates events and other calendar data.

To store the information in the google account, the user must register with a Google account and the user's authentication with google will be done through the Firebase platform. In order to use this platform, the application must have installed Firebase SDK that will provide us with the authentication service through Google (FirebaseAuth).



## 2.2 High level components and their interaction

The overview scheme presented in the last point naturally leads to the high level components view. The application is just composed by three tiers, the customer mobile application, the server we use to access and compute some information (Google server) and the database to storage some information about each user as the travel preferences. All the information related to the calendar is storage in the Google account of the user.



## Captura%20de%20pantalla%202017-11-09%20a%20las%2018.41.18.png2.3 Component view

- Customer: the customer´s smartphone application.

- ApplicationService: interface through which the user and the system communicate. For this it is necessary that the user has activated both the geolocation and the internet on his smartphone.

- Router: routes the requests to the appropriate system component.

- LoginController: provides the user's information to be able to log him with an authorized Google account and thus be able to use the resources offered by the Google APIs.

- LocatorHelper: manages the requests of locating a customer or a car by means of GoogleMapsGateway.

- RouteHelper: manages the requests to show the user the selected route to reach the destination.

- GoogleMapsGateway: Google Maps API providing the services which enable the system to locate devices and convert a string of text provided by the customer into a location.

- MapsInterface : Internet interface providing a standard HTTPS requests handler.

- Notificationhelper: manage notifications.

- Model: representation of the world and the information with which the system operates.

- FirebaseAuth: requires the Google email of the user to load his calendar.

- PushGateway: manage push notifications.

- CalendarController: provides information about the calendar of the user to the GoogleCalendar to I DON’T KNOW BECAUSE THE GOOGLE CALENDAR GATEWAY MUST PROVIDE INFORMATION ABOUT THE CALENDAR OF THE USER BUT ALSO THE CALENDAR CONTROLLER MUST PROVIDE INFORMATION TO THE GOOGLE CALENDAR TO UPDATE THE INFORMATION.

- UserController: requires the information about the location of the user and get the information about the user travelPreferences so that the system takes it into account when offering the user the routes.

## 2.4 DEPLOYMENT VIEW

The system shall adhere to certain requirements regarding reliability and performance. Those will be discussed in greater detail in section 3.

### 2.4.1. Regulatory policies

Travlendar does not use a database and customers do not need to deliver sensitive information. Travlendar simply provides the means customers use to manage their events locally on their device.

### 2.4.2 Hardware constraints

- The app should run on an Android 4.4+ or iOS 9.0+ smartphone.

- Internet connection should be enabled.

- Geolocation should be enabled.

- The app should run in portrait mode.

- The app should run in landscape mode.

## 2.5 Assumptions and dependencies

For Travlendar to function in a correct manner, these assumptions and dependencies must be met:

[D1] The smartphone device has constant internet connectivity.

[D2] The smartphone device has geolocation turned on at all times.

[D3] The phone is synchronized correctly with the current time of the user's region via mobile link or manually.

[D4] If for any reason the user has not attended a meeting, the system will give indications to the next meeting from the current location of the user.

[D5] User correctly enters meeting address when created.

# 3. Specific requirements

## 3.1 External interface requirements

### User Interfaces

This part of the document presents the application interface. The application is divided mainly in two views by a tap bar controller. In the main one the user activates or deactivates the means of transport they want. The second view presents the list with the events we have on the selected day. The user can change the day by browsing the calendar at the top of the view. Also in this view there is a button in the top right to add a meeting.

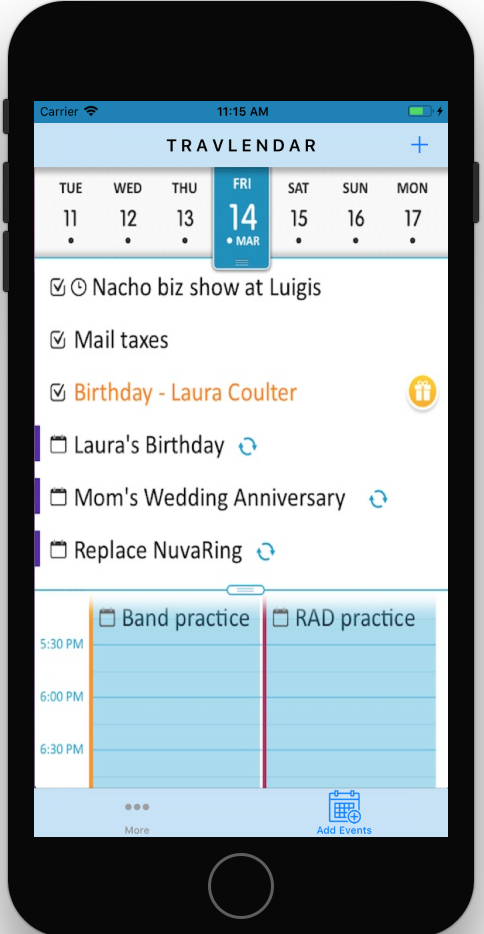


Figure 2: A list of events at a specific date

Figure 1: Main Travlendar screen

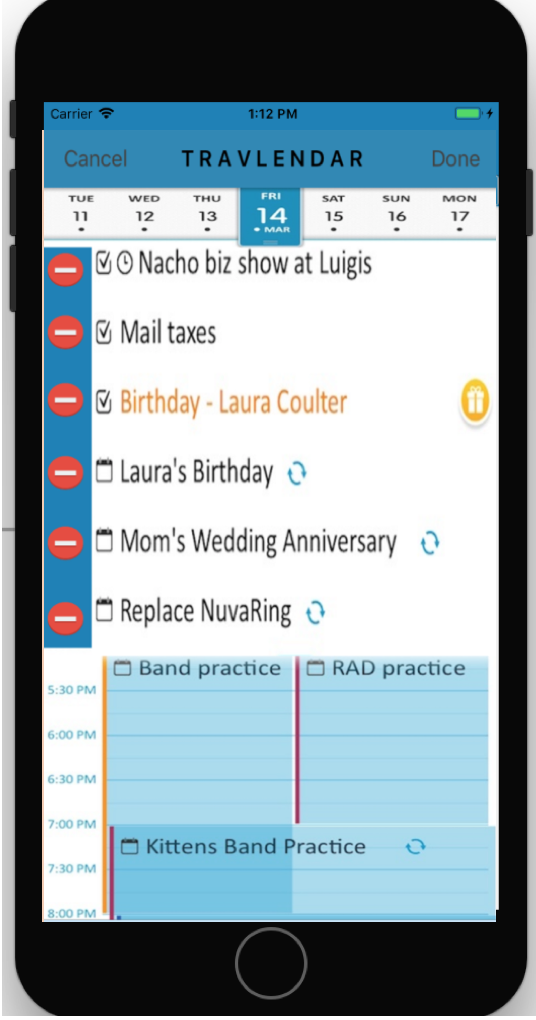


Figure 4: Deleting events

Figure 3: Adding an event

## 3.2 Functional requirements

This section specifies the requirements of each goal previously named.

We are using the MoSCoW model for the priority of the requirements. According to this model, requirements can have four diferent priorities, namely: must have, should have, could have and won’t have.

* [G1] Allow a User to manage appointments:

[R1] The system must allow the user to place an event in his agenda.

[R2] The system must allow the user to add a title to his meeting.

[R3] The system must allow the user to add a start and an end location to his meeting.

[R4] The system must allow the user to add the start and the end time of his meeting.

[R5] The system must detect the current location of the mobile app user.

[R6] The system must calculate the estimated travel time to reach a meeting.

[R7] The system must notify the user if there is an event (including travel time) overlap.

[R9] The system must be able to obtain different routes between the events using different means of transport in such a way that the user has several options to choose from.

* [G2] Allow a User to specify their own travel preferences:

[R8] The system must allow the user to activate and deactivate preferred means of transport.

[R9] The system must be able to obtain different routes between the events using different means of transport in such a way that the user has several options to choose from.

* [G3] Allow a User to introduce the breaks that he requires during the day and the temporal range in which he wants to do the rest:

[R10] The system must allow the user to place a break in his agenda.

[R11] The system must allow the user to add the time interval when the break should take place.

[R12] The system must allow the user to add the duration of his break.

[R7] The system must notify the user if there is an event (including travel time) overlap.

* [G4] Users should receive alerts

[R7] The system must notify the user if there is an event (including travel time) overlap.

[R13] The system should be able to check the weather at given location.

[R14] The system should be able to check the availability of transport at given location.

[R15] Users must receive an alert if the weather conditions are not pleasant when using means of transport such as walking or cycling.

[R16] Users must receive an alert if a certain way of transport to their appointment is not available.

[R17] Users must receive an alert 15 minutes before they have to leave for their next meeting.

* [G5] The user should receive directions from his current location to the location of the event:

[R5] The system must detect the current location of the mobile app user.

[R6] The system must calculate the estimated travel time to reach a meeting.

[R18] The system must assist the user in his travels and display his current and target location while he is travelling towards an event.

* [G6] The system must consider days in which the public transport or the transport chosen by the user is not available or delayed:

[R14] The system should be able to check the availability of transport at given location.

* [G7] The system must consider the weather when an event takes place

[R13] The system should be able to check the weather at given location.

## 3.3 Performance requirements

[P1] The content of a page in the mobile app must load within 5 seconds.

[P2] The content of a page in the mobile app should load within 2 seconds.

[P3] The content of a page in the mobile app could load within 1 second.

[P4] An external input to the system must result in visual feedback within 1 second.

[P5] An external input to the system should result in visual feedback within 0.5 seconds.

[P6] An external input to the system could result in visual feedback within 0.2 seconds.

## 3.4 Design Constraints

These are discussed in detail in section 2.4 of this document.

## 3.5 Software System Attributes

### Reliability

The probability of failure of the system should be close to 0 at all times. This will require updates to improve the services of the application and correct possible errors of the early versions.

### Availability

Availability depends on the reliability of the system. As reliability increases, so does availability. The system should provide its services at all times.

### Security

The security does not take a very important value since the data is stored in the application itself, so it is very difficult for information theft to occur.

### Maintainability

It is one of the most important points that represents the majority of the expenses of the project. There must be continuous maintenance of the application to make it work in the best possible way.

### Portability

Most of the functionality of the app is provided by different APIs (Google maps, Google calendar, etc.). Since the app will be running both on Android and iOS, some of the software or the logic behind it will be reused.

# 4. Use cases

1. User creating a new appointment

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G1] |
| **INPUT CONDITIONS** | There are no entry conditions. |
| **EVENTS FLOW** | * The user should press the “Calendar” bottom. * The user should press the “+” button. * The user should enter a title for the event. * The user should enter a start time for the event. * The user should enter an end time for the event. * The user should enter his location before the event (start location). * The user should enter the location of the event (end location). * The user should press the “Save” button. |
| **OUTPUT CONDITIONS** | The appointment is added to the calendar. |
| **EXCEPTIONS** | 1. The user inserts a non-existing location.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 6.  2. The user inserts an event that overlaps with another one.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4. |

1. User editing an appointment

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G1] |
| **INPUT CONDITIONS** | The user must have created an event. |
| **EVENTS FLOW** | * The user should press the “Calendar” bottom. * The user should press the “Edit” bottom. * The user should press on the event he wishes to edit. * The user should edit the event. * The user should press the “Save” button. |
| **OUTPUT CONDITIONS** | The appointment is added to the calendar. |
| **EXCEPTIONS** | 1. The user inserts a non-existing location.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4.  2. The user inserts an event that overlaps with another one.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4. |

1. User deleting an appointment

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G1] |
| **INPUT CONDITIONS** | The user must have created an event. |
| **EVENTS FLOW** | * The user should press the “Calendar” bottom. * The user should press the “Edit” bottom. * The user should press on the red circle at the left of the event he wishes to delete. |
| **OUTPUT CONDITIONS** | The appointment is removed from the calendar. |
| **EXCEPTIONS** |  |

1. User creating a new break

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G3] |
| **INPUT CONDITIONS** | There are no entry conditions. |
| **EVENTS FLOW** | * The user should press the “Calendar”” bottom. * The user should press the “+” button. * The user should press the “New break” button * The user should enter a title of the break. * The user should enter the interval time where he/she wants to do the break. * The user should enter the duration of the break. * The user should press the “Save” button. |
| **OUTPUT CONDITIONS** | The break is added to the calendar. |
| **EXCEPTIONS** | 1. The user inserts a break that overlaps with another event.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 5. |

1. User editing a break

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G3] |
| **INPUT CONDITIONS** | The user should have created a break. |
| **EVENTS FLOW** | * The user should press the “Calendar”” bottom. * The user should press the “Edit” button. * The user should press on a break. * The user should edit the input fields of the break. * The user should press the “Save” button. |
| **OUTPUT CONDITIONS** | The break is added to the calendar. |
| **EXCEPTIONS** | 1. The user inserts a break that overlaps with another event.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 4. |

1. User deleting a break

|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G3] |
| **INPUT CONDITIONS** | The user should have created a break. |
| **EVENTS FLOW** | * The user should press the “Calendar”” bottom. * The user should press the “Edit” button. * The user should press on the red circle at the left of the event he wishes to delete. |
| **OUTPUT CONDITIONS** | The break is deleted from the calendar. |
| **EXCEPTIONS** |  |

1. User specifies travel preferences

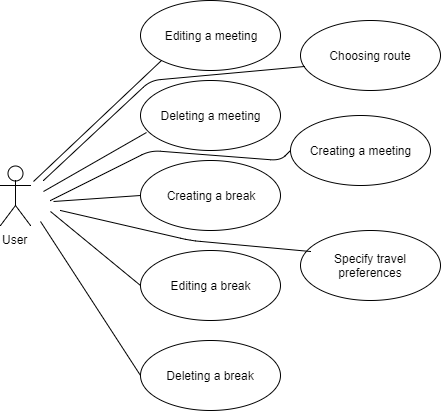
|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G2] |
| **INPUT CONDITIONS** | There are no entry conditions. |
| **EVENTS FLOW** | * The user should press the “More” button. * The user should choose his preferences by using the radio buttons mapped to the different means of travel. |
| **OUTPUT CONDITIONS** | The travel preferences have been chosen. |
| **EXCEPTIONS** | 1. The user deselects all travel preferences.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 2. |

1. User chooses route

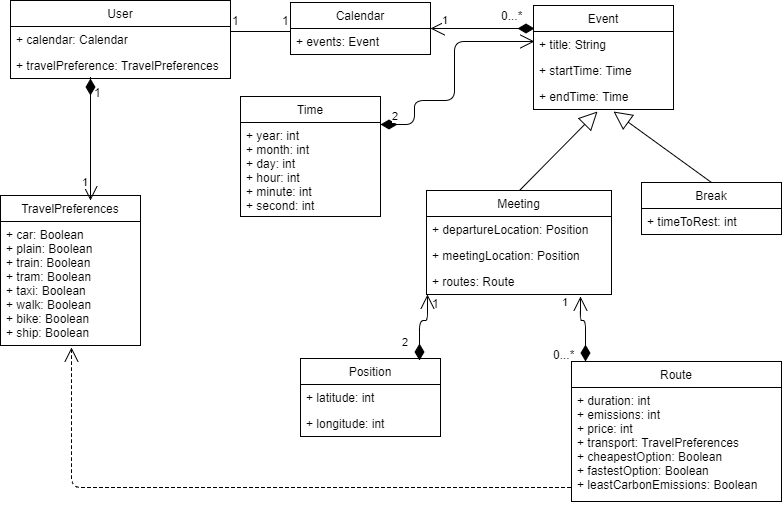
|  |  |
| --- | --- |
| **ACTORS** | User |
| **GOALS** | [G2] [G6] [G7] |
| **INPUT CONDITIONS** | The user should have created an appointment. |
| **EVENTS FLOW** | * The user should press the “Calendar”” bottom. * The user should press on an appointment. * The user should press the “Travel” button. * The user should choose one of the suggested routes. * The system will display a map with a marker which represents the user while he reaches his destination. |
| **OUTPUT CONDITIONS** | The route has been chosen. |
| **EXCEPTIONS** | 1. There is no route available.  This exception is handled notifying the issue to the User and taking back the Event Flow to the point 3. |

# 5. UML diagrams

Use case diagram:

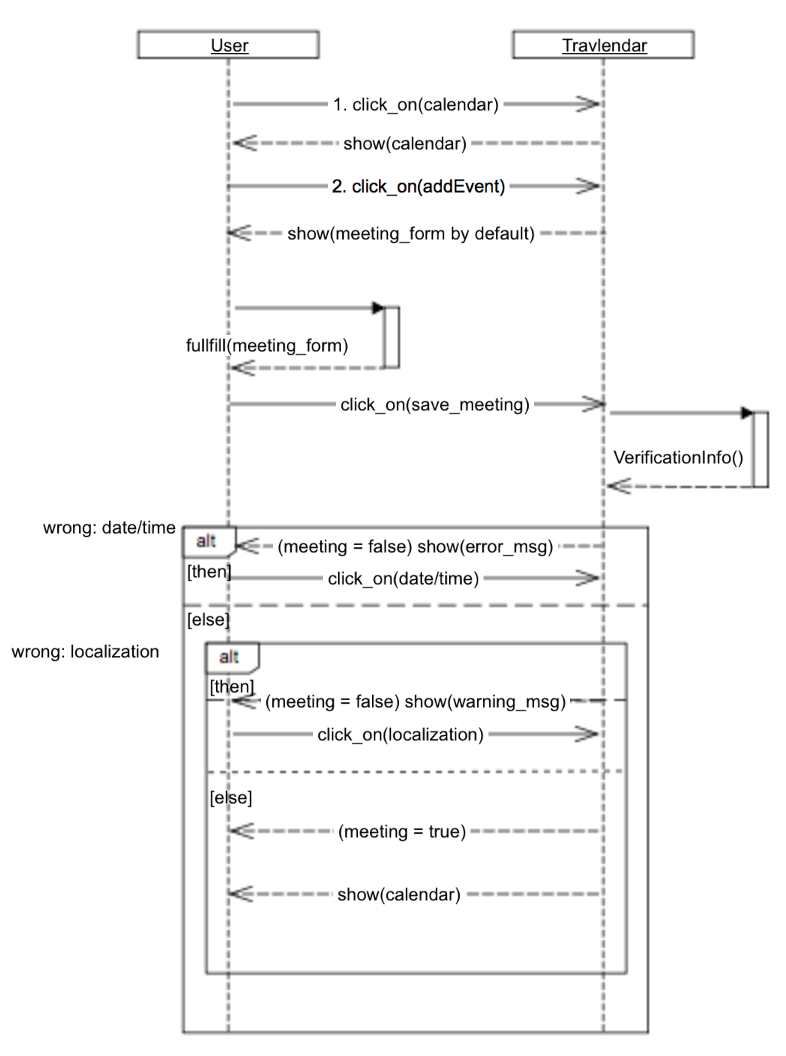


Class diagram:

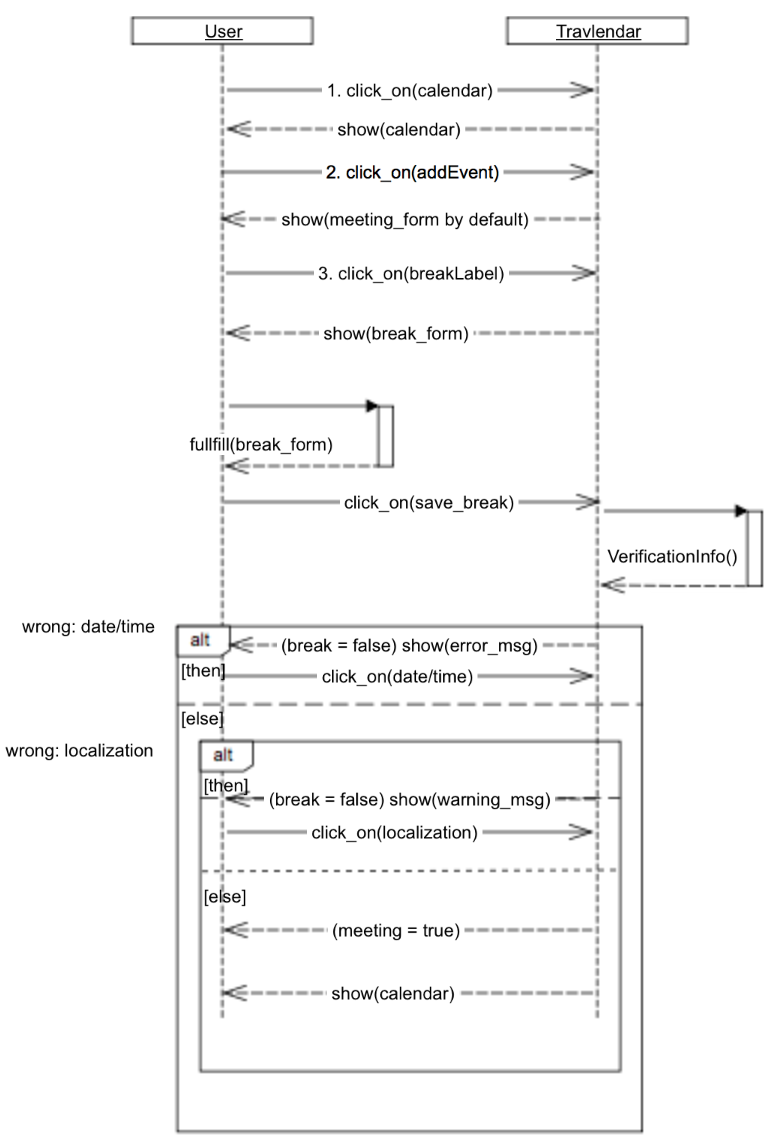


Sequence diagrams:

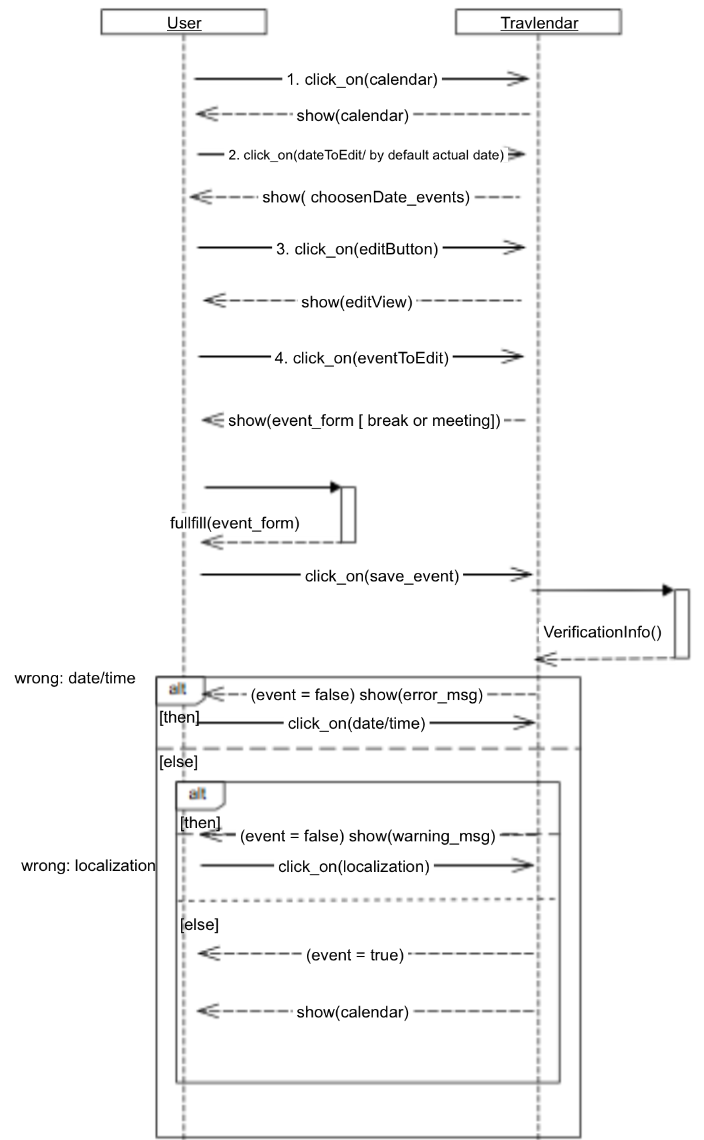
1. *Creating a meeting*



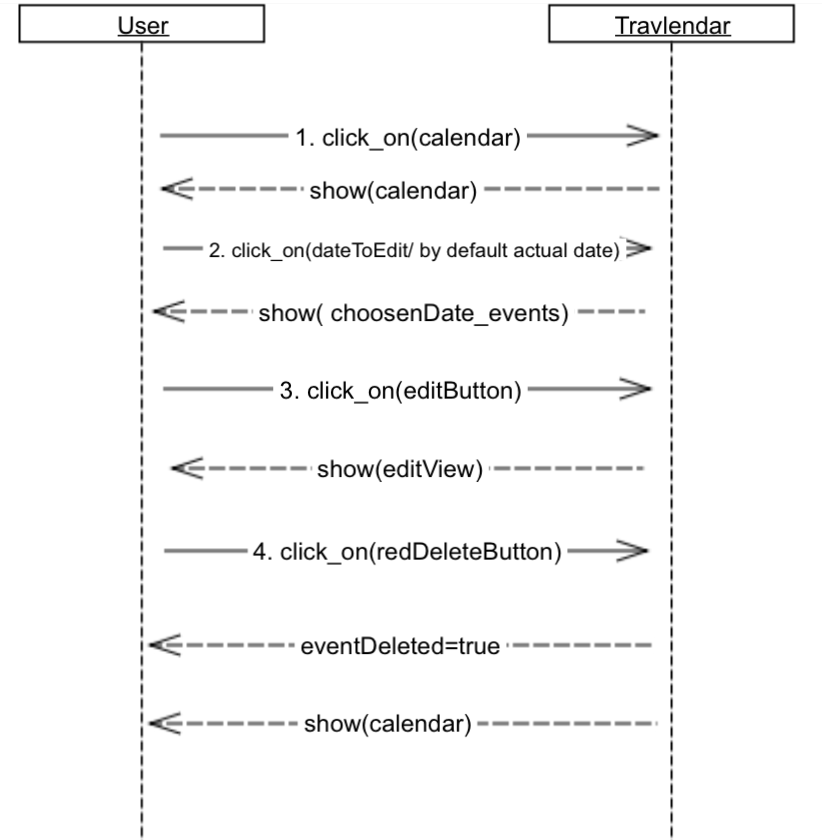
1. *Creating a break*

**

1. *Editing event*



1. *Deleting an event*

**

# 6. Formal analysis using Alloy

## 6.1 Purpose

We are using Alloy to model a User creating and deleting events. We can prove that creating an event does not overlap with other events and the data of the created event is complete. We can also show that removing an event successfully removes it from the calendar of the user.

## 6.2 Code

open util/boolean

open util/time

open util/integer

sig Calendar{

events: set Event

}

sig User{

travellingPreferences: one TravelPreferences,

calendar: one Calendar

}

sig Route{

duration: one Int,

transport: one TravelPreferences,

cheapestOption: one Bool,

fastestOption: one Bool,

leastCarbonEmissions: one Bool,

emissions: one Int,

price: one Int

}

{

duration>0

price>0

emissions>0

}

sig TravelPreferences{

car: one Bool,

plain: one Bool,

train: one Bool,

tram: one Bool,

taxi: one Bool,

walk: one Bool,

bike: one Bool,

ship: one Bool

}

abstract sig Event{

title: one String,

startTime: one Time,

endTime: one Time

}

sig Meeting extends Event{

departureLocation: one Position,

meetingLocation: one Position,

routes: set Route

}

sig Break extends Event{

timeToRest: one Int

}{

timeToRest>0

}

sig Position{

latitude: one Int,

longitude: one Int

}

--Each route has only one type of transport. It is true that they might have more but it is assumed for simplicity.

fact justOneTransportRoute{

all r: Route, u: User | r.transport.car.isTrue && u.travellingPreferences.car.isTrue => not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.ship.isTrue && u.travellingPreferences.ship.isTrue=> not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.car.isTrue

all r: Route, u: User | r.transport.train.isTrue && u.travellingPreferences.train.isTrue => not r.transport.bike.isTrue && not r.transport.car.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.bike.isTrue && u.travellingPreferences.bike.isTrue => not r.transport.car.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.plain.isTrue && u.travellingPreferences.plain.isTrue => not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.car.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.walk.isTrue && u.travellingPreferences.walk.isTrue => not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.car.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.taxi.isTrue && u.travellingPreferences.taxi.isTrue => not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.tram.isTrue && not r.transport.plain.isTrue && not r.transport.car.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

all r: Route, u: User | r.transport.tram.isTrue && u.travellingPreferences.tram.isTrue=> not r.transport.bike.isTrue && not r.transport.train.isTrue && not r.transport.car.isTrue && not r.transport.plain.isTrue && not r.transport.taxi.isTrue && not r.transport.walk.isTrue && not r.transport.ship.isTrue

}

--Just one option of cheapest, fastest and least carbon emissions for each set of routes.

fact oneRouteSelected{

all r1,r2: Route | r1.duration < r2.duration =>r1.fastestOption.isTrue && not r2.fastestOption.isTrue

all r1,r2: Route | r1.emissions < r2.emissions =>r1.leastCarbonEmissions.isTrue && not r2.leastCarbonEmissions.isTrue

all r1,r2: Route | r1.price < r2.price =>r1.cheapestOption.isTrue && not r2.cheapestOption.isTrue

}

--Each event has his own title.

fact titleIsUnique{

all disj e1,e2: Event | disj[e1.title,e2.title]

}

--The latitude and longitude must be inside the valid values.

fact validPosition{

all p:Position | gt[90, p.latitude] && gt[p.latitude, -90] && gt[180, p.longitude] && gt[p.longitude, -180]

}

--At any Time there must be just one event. Also the startTime and the endTime of the event must be different

fact oneEventAtOneTime{

all disj e1,e2: Event | (gt[e1.startTime,e2.endTime] or gt[e2.startTime,e1.endTime])

all disj e3,e4: Event | disj [e3.endTime, e4.endTime] and disj[e3.startTime, e4.startTime]

all disj e: Event | disj[e.startTime, e.endTime]

}

--Each User must have his own calendar

fact unicCalendar{

all disj u1,u2: User | disj[u1.calendar, u2.calendar]

}

--There must be at leat one travel preference enable because if not, is not possible to show any route to the user

fact atLeastOneTravelPreferenceEnabled{

all u: User | (u.travellingPreferences.car.isTrue or u.travellingPreferences.bike.isTrue or u.travellingPreferences.car.isTrue or u.travellingPreferences.ship.isTrue or u.travellingPreferences.train.isTrue or

u.travellingPreferences.walk.isTrue or u.travellingPreferences.tram.isTrue or u.travellingPreferences.taxi.isTrue or u.travellingPreferences.train.isTrue or u.travellingPreferences.plain.isTrue)

}

-- Checks if the data of the event is not null.

pred isDataCompleted[e: Event]{

e.title != none

e.startTime != none

e.endTime != none

}

--Checks if in the interval of time t1:startTime, t2:endTime of the event there is not any events in the

--calendar of the user.

pred freeTimeWindow[t1,t2: Time, c: Calendar]{

all e: c.events | gt[t1, e.endTime] or gt[e.startTime, t2]

}

--Adding an event to the set of events of the calendar of the user u. After that there is created

--a new user u' with the new event included. Fist of all, checks that the event is not in the set.

--After that is added and finally checks that is inside the set after adding it.

pred addEvent[u,u': User, e: Event]{

not e in u.calendar.events

u'.calendar.events = u.calendar.events + e

e in u'.calendar.events

}

--Removing an event to the set of events of the calendar of the user u. After that there is created

--a new user u' without the event e. Fist of all, checks that the event is in the set.

--After that is removed and finally checks that is not inside the set after removing it.

pred removeEvent[u,u': User, e: Event]{

e in u.calendar.events

u'.calendar.events = u.calendar.events - e

not e in u'.calendar.events

}

--Before creating the new event, it is checked if the event has a title and also if there is a free window

--in the interval of time of the event in the calendar of the user.

pred createNewEvent[e: Event, u,u': User]{

isDataCompleted[e]

freeTimeWindow[e.startTime, e.endTime, u.calendar]

addEvent[u,u',e]

}

--After removing the event, it is checked if there is a free window in the calendar of the user in the interval

--time of the event removed.

pred deleteEvent[e: Event, u,u': User]{

removeEvent[u,u',e]

freeTimeWindow[e.startTime, e.endTime, u'.calendar]

}

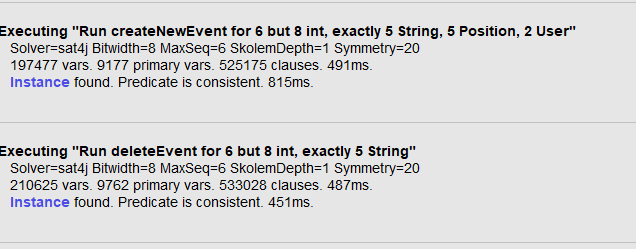
run createNewEvent for 6 but exactly 5 String, 5 Position, 2 User, 8 int

run deleteEvent for 6 but 8 int, exactly 5 String

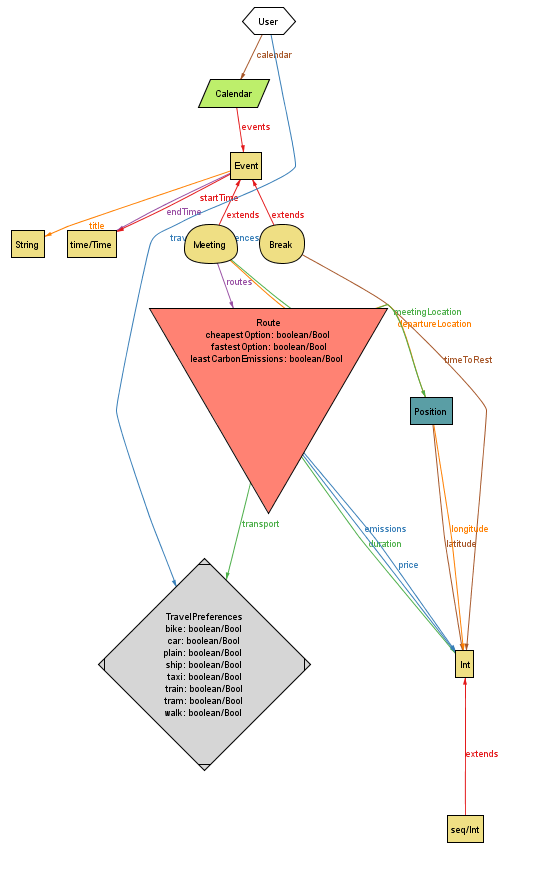
## 6.3 Results



### 6.3.1 Proof of consistency



### 6.3.2 Generated world



# 7. Effort spent

|  |  |  |
| --- | --- | --- |
| Name | Date | Hours spend |
| Plamen | 12/10/17 | 2 |
| Plamen | 15/10/17 | 5.5 |
| Plamen | 16/10/17 | 4 |
| Plamen | 17/10/17 | 1.5 |
| Plamen | 19/10/17 | 4 |
| Plamen | 21/10/17 | 5 |
| Plamen | 22/10/17 | 2 |
| Plamen | 24/10/17 | 6 |
| Plamen | 27/10/17 | 1 |
| Victor | 12/10/17 | 2 |
| Victor | 14/10/17 | 7 |
| Victor | 15/10/17 | 2 |
| Victor | 16/10/17 | 4 |
| Victor | 19/10/17 | 7 |
| Victor | 21/10/17 | 3 |
| Victor | 22/10/17 | 3.5 |
| Victor | 24/10/17 | 3 |

Total: Plamen: 31 hours

Victor:31.5 hours

# 8. References