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

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 or in other words to 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. Google maps API:
   - https://developers.google.com/maps/web-services/?hl=es-419
   - https://github.com/googlemaps/google-maps-services-java

2. Google Calendar API:
   - https://fullcalendar.io/docs/google_calendar/
   - https://developers.google.com/google-apps/calendar/overview

3. Yahoo Weather API:
   - https://developer.yahoo.com/weather/
   - https://github.com/fedy2/yahoo-weather-java-api

4. Beep slides.

5. Sequence Diagrams:
   - https://www.draw.io

6. User Interface:
   - https://developer.apple.com/xcode/

7. Calendar Notifications:
   - https://developers.google.com/google-apps/calendar/v3/push

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 of this document we are going to discuss the architectural design of the system and more specifically the different components and how they interact with eachother.

3. In the third part of the document we are going to discuss some of our most relevant algorithms developed with the use of JEE7.

4. In the forth part of the document we are going to discuss how the user interface of the system will look like.

5. In the fifth section of the document we are going to trace requirements and provide information about in which component they are implemented.

6. The sixth part of the document is dedicated to the implementation, integration and test plan which we are going to discuss in detail.

7. The last part of this document is about how much effort our team spend and what references we used.

2. ARCHITECTURAL DESIGN

2.1 OVERVIEW

In our system we only have one element - the user application. It will serve as an intermediary between the user and other systems to perform certain queries.

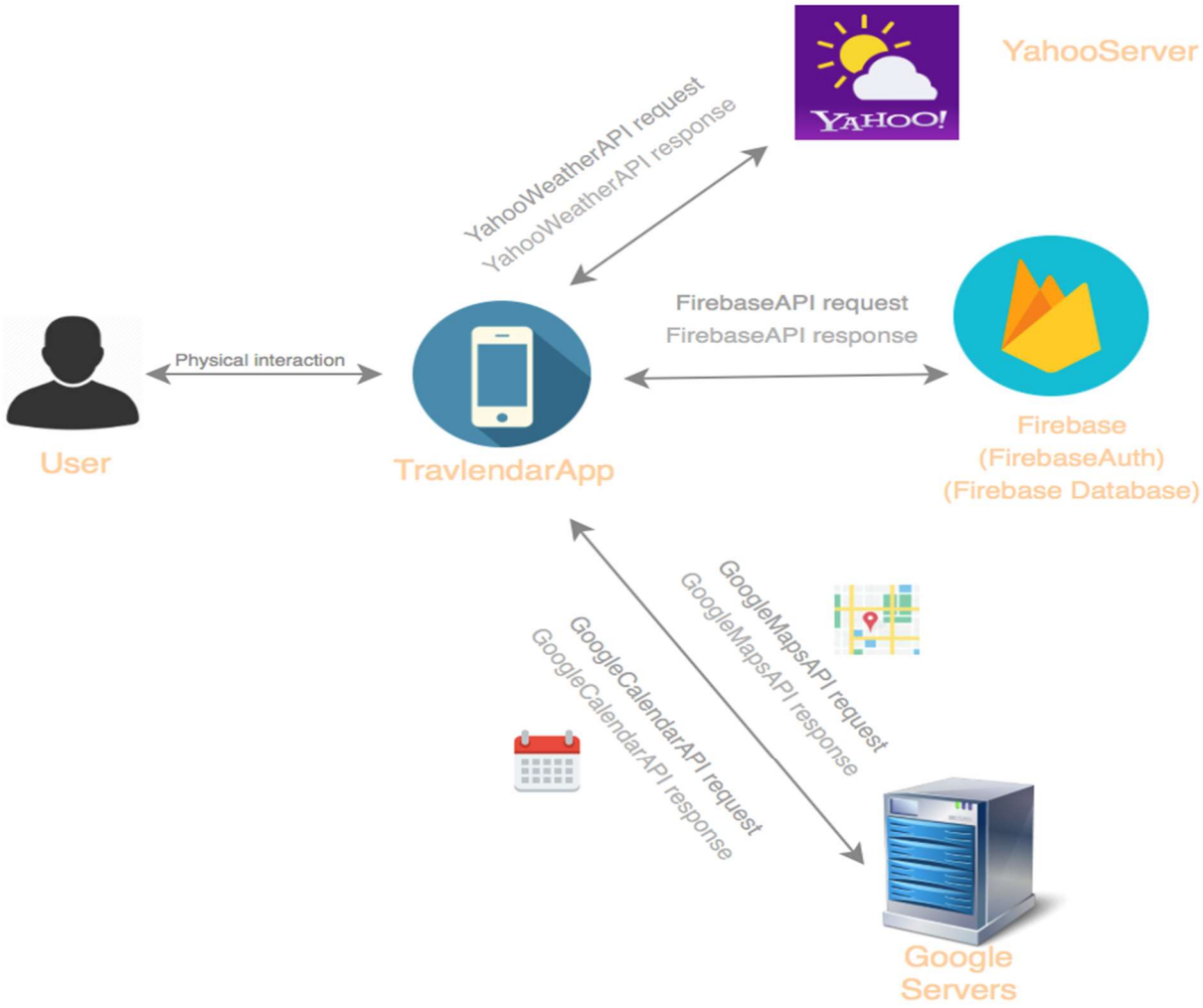
1) User application: represents the user who has downloaded the Travlendar application. The application allows the user to manipulate their own calendar in such a way that they can create, edit and delete events. Within the events, the user will have to specify when creating a new one if this is a break or a meeting. 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 information of the events will be stored in the user's Google Calendar. In addition to this we will use the database provided by Firebase that will allow us to store some variables of the user as well as their travel preferences. For this reason, the user must have a google account with which he must log in. We can avoid the purchase and maintenance of a server using 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 all over again.

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

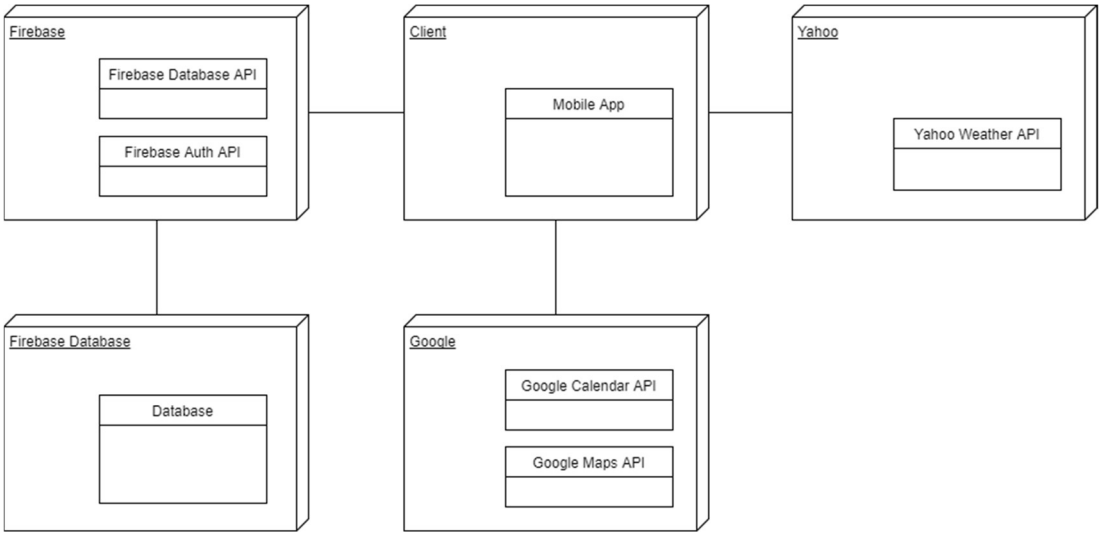
- Google Maps: representation of the Google Maps Web Services. In particular:
  - GeoCoding API for the conversion of a string of text into a GPS position.
  - GeoLocation API for the GPS localization of devices.
- Google Calendar API: manipulates events and other calendar data.
- Yahoo Weather API: provides information about different weather variables such as wind speed, barometric variables, pressure etc.
-Firebase Authentication API: Allows the communication between the user application and the firebase system and more specifically: the authentication functionality that it offers.
-Firebase Database API: Allows the communication between the user application and the firebase system and more specifically: the database functionality that it offers.

To store the information in google calendar, the user must log-in 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).

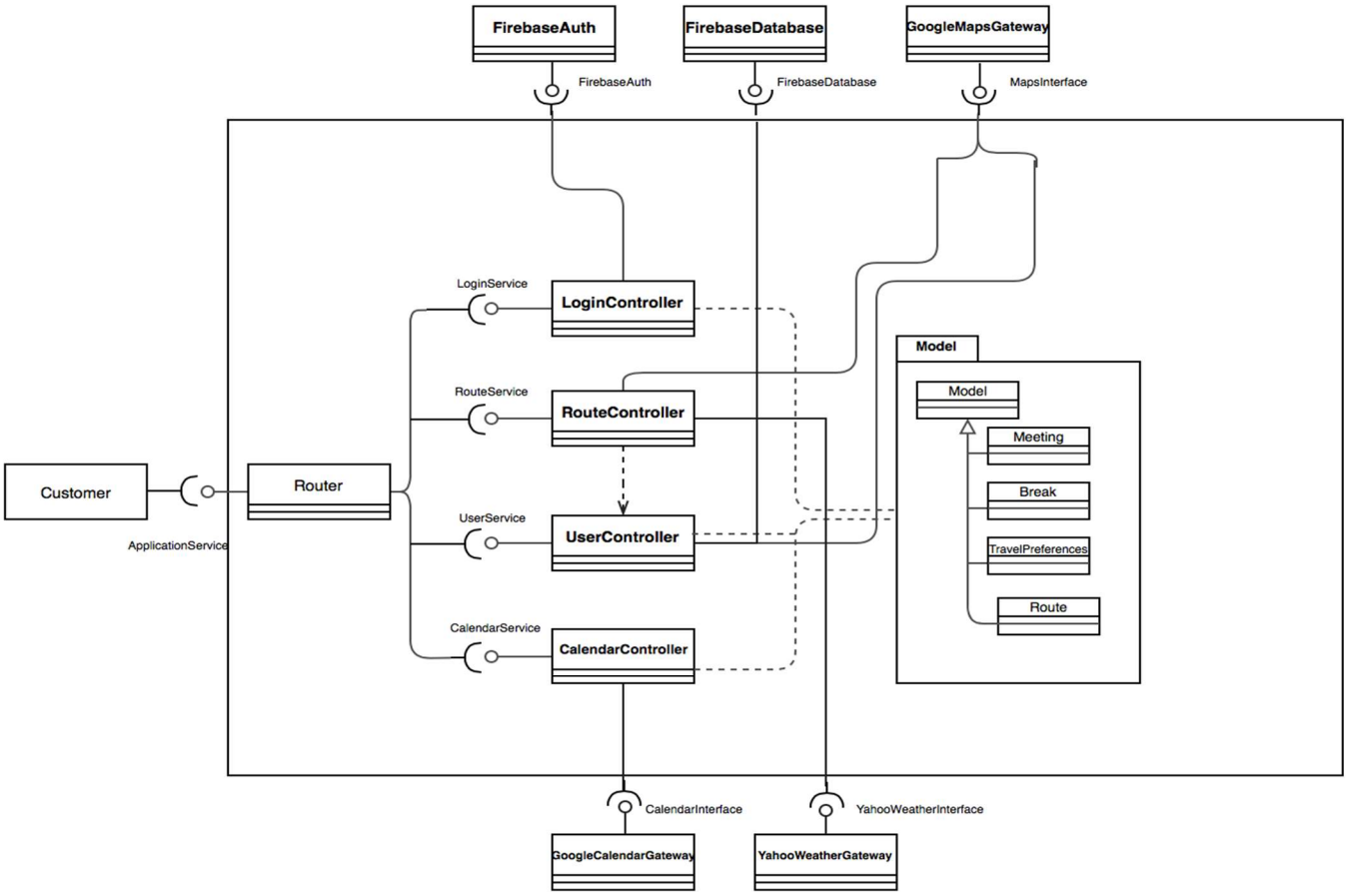


2.2 HIGH LEVEL COMPONENTS AND THEIR INTERACTION

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.



2.3 COMPONENT VIEW



- 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.

- Customer: the customer´s smartphone application.

- LoginController: requires the resources of FirebaseAuth to authenticate the user correctly with his Google account. Also provides the user a GoogleLogin token to access the calendar resources.

- FirebaseAuth: provides the resources for user authentication.

-RouteController: requires information about the possible routes to reach the destination to the Google API and provides the router and then the user with a route to reach their destination according to the travel preferences of the user. Also the RouteController gets the location of the user through the UserController.

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

- 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.

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

- CalendarController: requires the services of the Google Calendar API and provides the calendar of the user to the router. To access to the Google Calendar the user must have previously logged in.

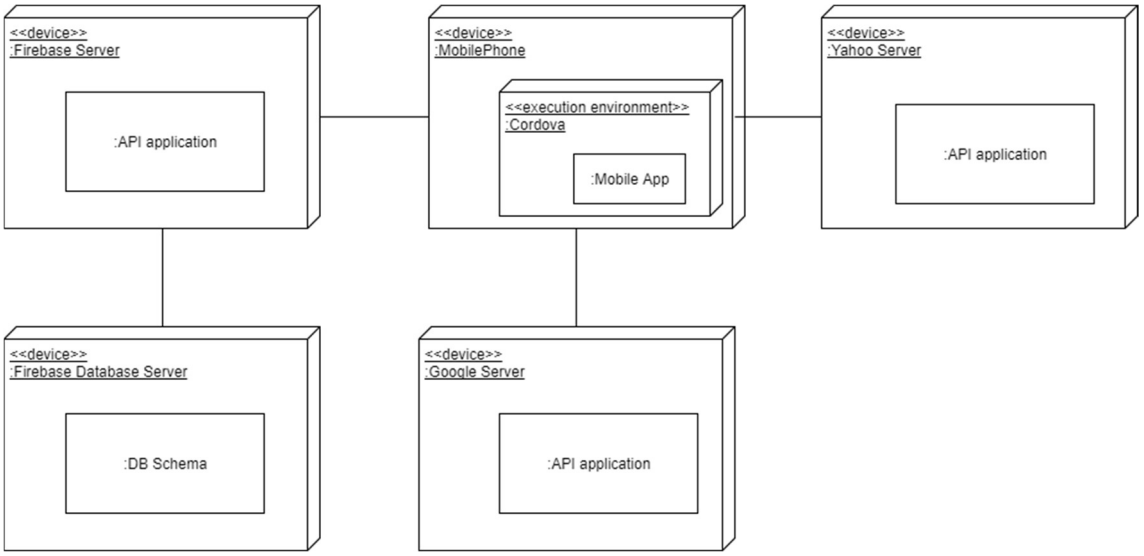
- GoogleCalendarGateway: Google Calendar API providing the services which enable the system to access and modify the private calendar of the user.

- FirebaseDatabase: provides the resources to store information about the user.

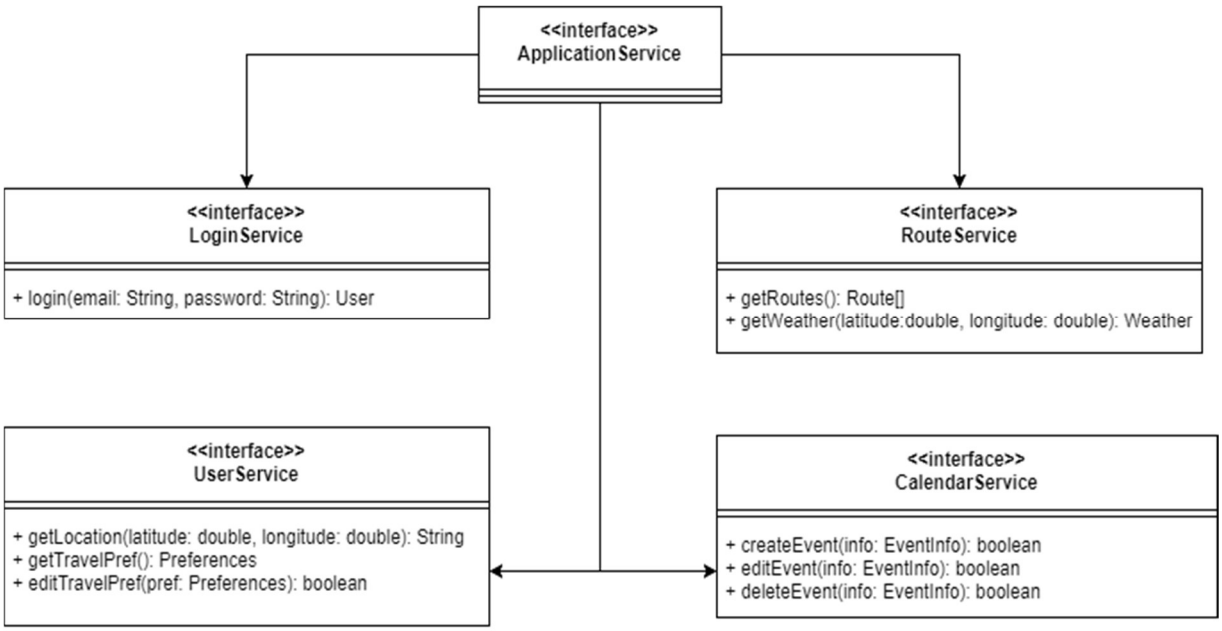
- YahooWeatherGateway: Yahoo Weather API providing services which enable the system to get access to the weather at the location of the user to inform him about what routes the system does not recommend.

The notifications that the application wants to give to the user will be carried out by the Google Calendar API.

2.4 DEPLOYMENT VIEW

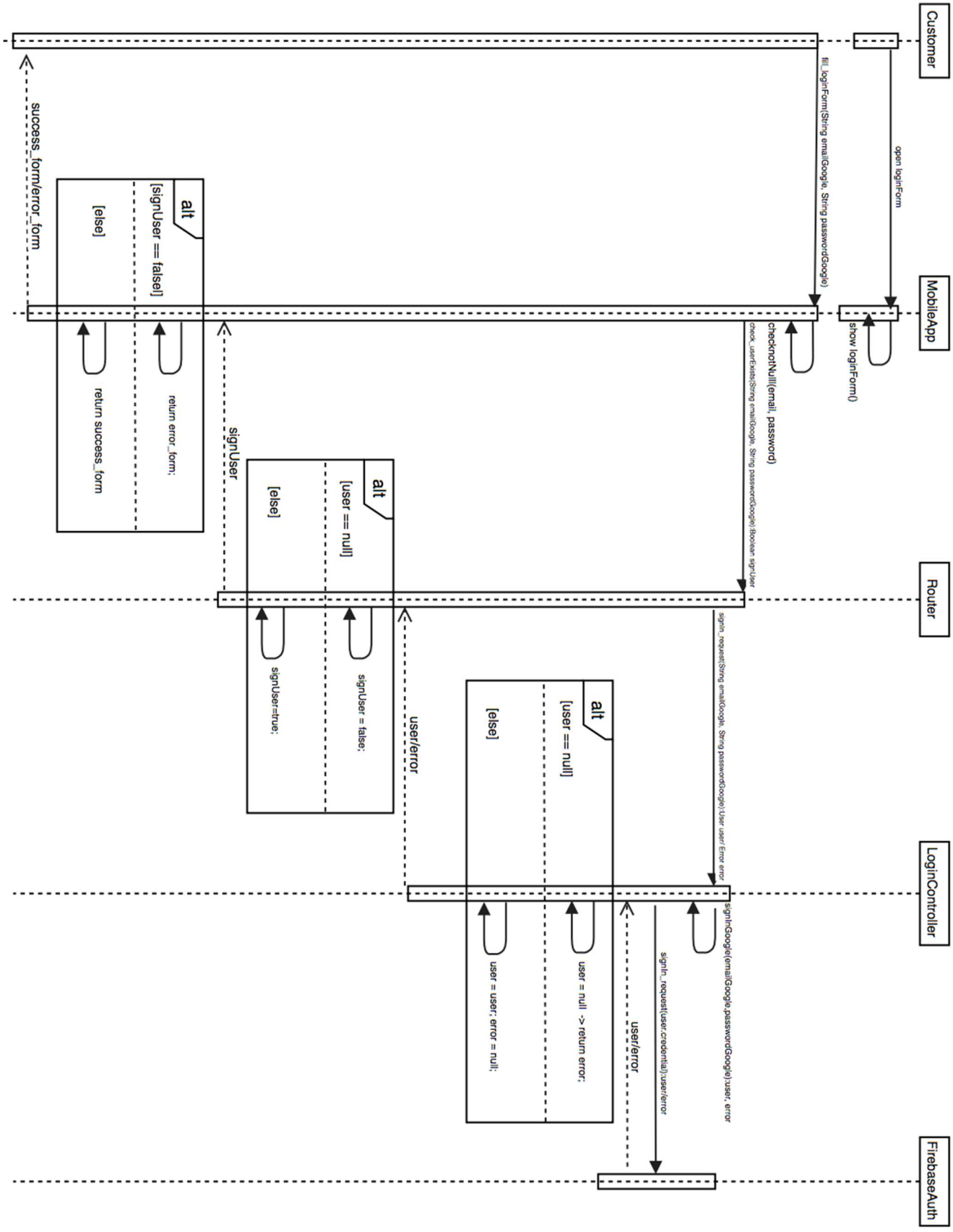


2.5 COMPONENT INTERFACES

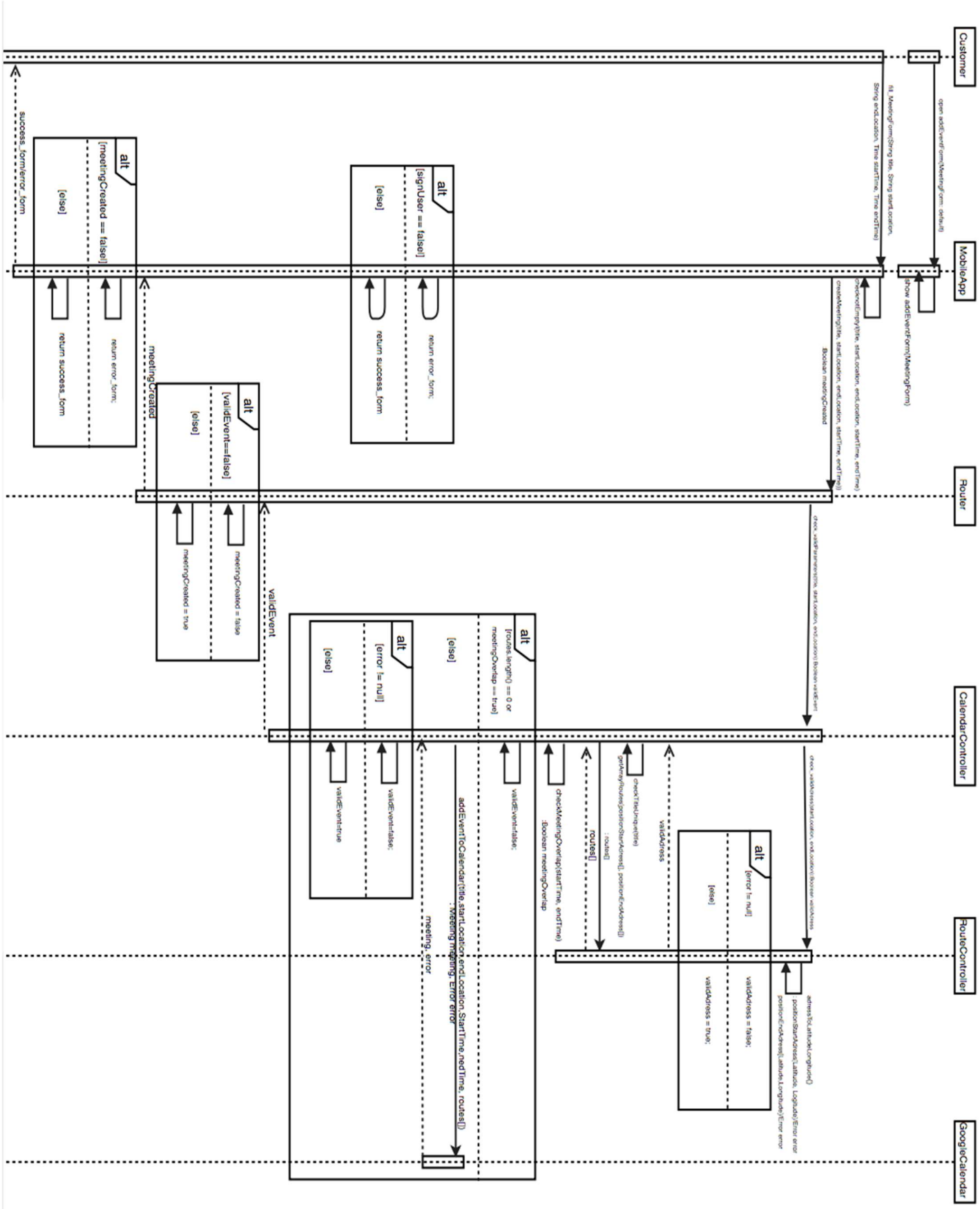


2.6 RUNTIME VIEW

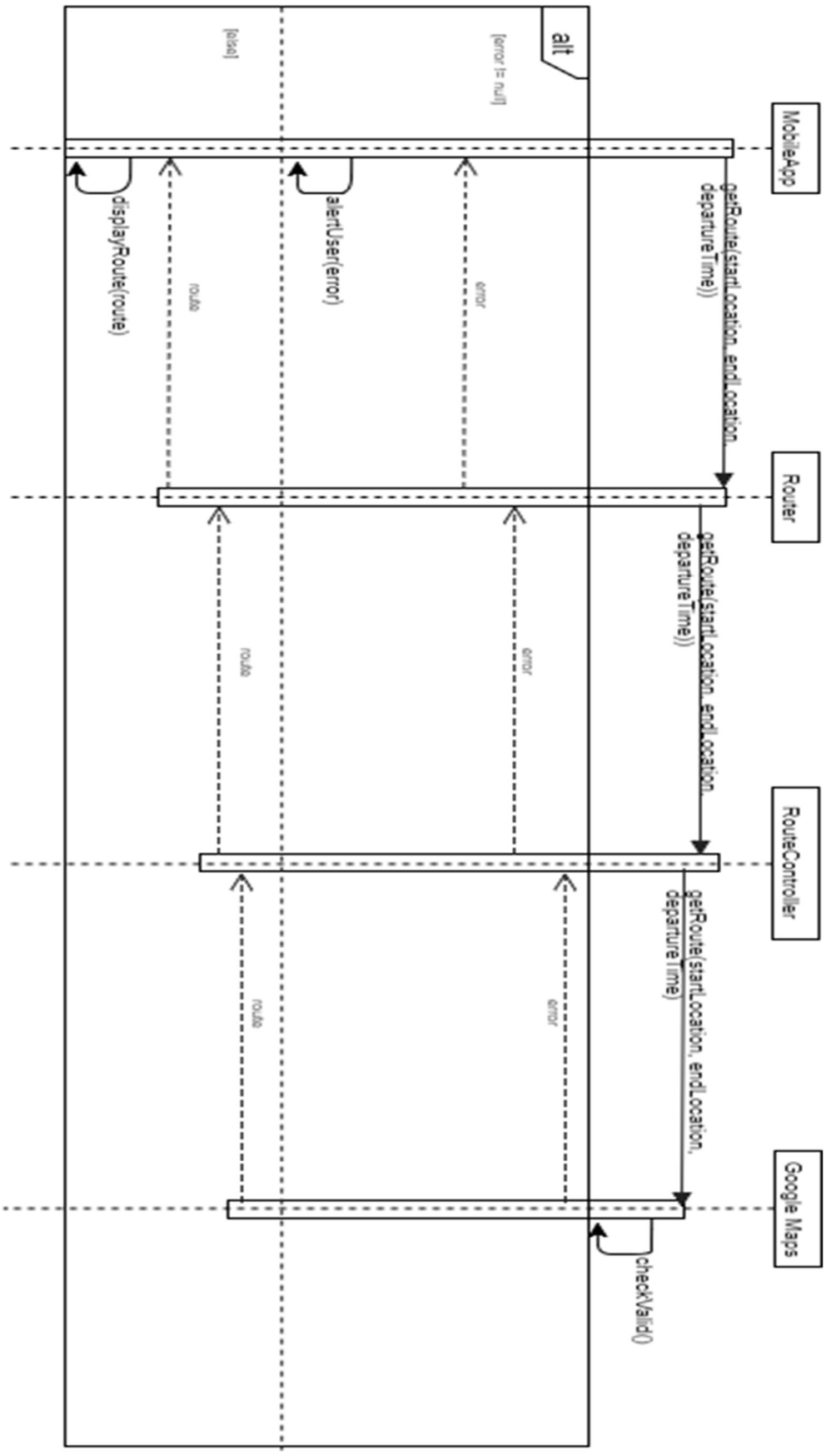
1) Log In Sequence Diagram:



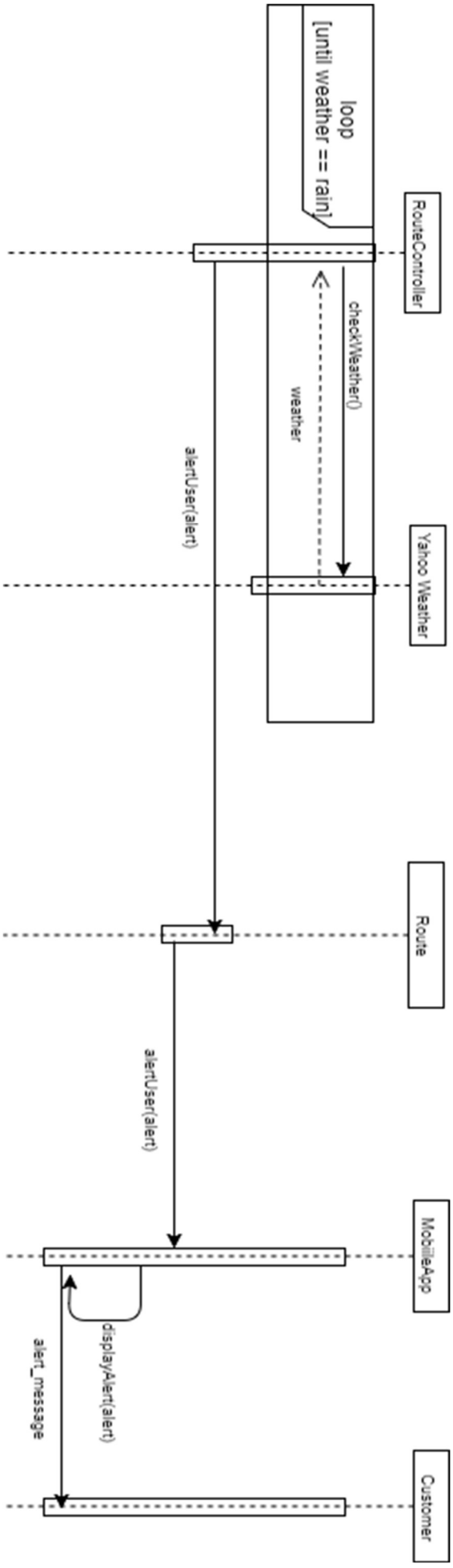
2) Add Meeting to a Calendar Sequence Diagram:



3) Get the route between two locations sequence diagram:



4) Check Weather sequence diagram:



You can see the sequence diagrams with better quality in these links:

LogIn:
https://github.com/VictorAlvaro/PaslievAlvaro/blob/master/LogInUserSequenceDiagram.png

AddMeeting:
https://github.com/VictorAlvaro/PaslievAlvaro/blob/master/addEventToCalendarSequenceDiagram.png

Get route:

https://github.com/VictorAlvaro/PaslievAlvaro/blob/master/sequenceDiagramgetroute.png

Check weather:

https://github.com/VictorAlvaro/PaslievAlvaro/blob/master/checkweather.png

3.ALGORITHM DESIGN

3.1 getMeeting:

Function that returns the meeting of the database with that title. If there is none with that title, return a null. For this we create a query of the entity manager that obtains a single result with the meeting with that title.

Input: Title of the meeting.

Output: Meeting object or null.

```
public Meeting getMeeting(String title) {
    try {
        return (Meeting) this.em.createQuery("SELECT m FROM Meeting m WHERE m.title=:title")
                .setParameter("title", title).getSingleResult();
    } catch (NoResultException ex) {
        return null;
    }
}
```

3.2 checkTimeSlot:

Function in charge of verifying if there is a free window in that space of time. For this, it receives two corresponding parameters with the time instantaneous variables start time of the meeting and end time of the meeting. First we get an ArrayList with all the events in the database from the private function getEvents (). Later we verify if in that space of time there is no other event in the user's calendar. If this is the case, the function will return a true. Otherwise, the answer will be false.

Inputs: start time of the meeting and end time of the meeting

Outputs: Boolean true or false.

```
private Boolean checkTimeSlot(ReadWritableInstant startTime, ReadWritableInstant endTime){
  ArrayList <Event> items = getEvents();
  for (Event event : items) {
    if(event.getEndTime().isAfter(startTime) && startTime.isAfter(event.getStartTime())){
        return false;
    }
    if(event.getEndTime().isAfter(endTime) && endTime.isAfter(event.getStartTime())){
        return false;
    }
  }
  return true;
}
```

3.3 getRoutes:

Private function to obtain an ArrayList with the possible routes to 4 different means of transport. For the development of this function we have used the Google Maps API. In the first place we will have a GeoApiContext that is the beginning to carry out requests to the services of the Google API. Later we check if the current user has one of the four transport modes activated and if it is activated, the routes with this means of transport are obtained. To obtain the directions to arrive at the destination, a request is made to the Api of Directions with origin departureLocation, destination meetinglocation, the time at which the meeting begins in order to arrive on time and the corresponding transport mode. In addition to this we obtain the time in seconds that is used to reach the destination that will be useful later to obtain the fastest route.

Input: departure location, meeting location and end time of the meeting.

Output: ArrayList with the directions to reach the meeting for activated means of transport.

```
private ArrayList<DirectionsResult> getRoutes(String departureLocation, String meetingLocation, ReadableInstant endTime) throws ApiException, InterruptedException, IOException {
    GeoApiContext context;
    User u = getCurrentUser();
    ArrayList<DirectionsResult> routes = null;
    context = new GeoApiContext.Builder().build();
    if (u.travelPref.getPreferences()[0] == true) {
        DirectionsApiRequest bycar = DirectionsApi.newRequest(context).destination(meetingLocation).origin(departureLocation).mode(TravelMode.DRIVING).arrivalTime(endTime);
        DirectionsResult routecar = bycar.await();
        DistanceMatrixApiRequest durationCarRequest = DistanceMatrixApi.newRequest(context).origins(departureLocation).destinations(meetingLocation).mode(TravelMode.DRIVING);
        DistanceMatrix matrix = durationCarRequest.await();
        distances.add(matrix.rows[0].elements[0].duration.inSeconds);
        routes.add(routecar);
    }
    if (u.travelPref.getPreferences()[1] == true) {
        DirectionsApiRequest bybike = DirectionsApi.newRequest(context).destination(meetingLocation).origin(departureLocation).mode(TravelMode.BICYCLING).arrivalTime(endTime);
        DirectionsResult routebike = bybike.await();
        DistanceMatrixApiRequest durationBikeRequest = DistanceMatrixApi.newRequest(context).origins(departureLocation).destinations(meetingLocation).mode(TravelMode.BICYCLING);
        DistanceMatrix matrix = durationBikeRequest.await();
        distances.add(matrix.rows[0].elements[0].duration.inSeconds);
        routes.add(routebike);
    }

    if (u.travelPref.getPreferences()[2] == true) {
        DirectionsApiRequest bywalk = DirectionsApi.newRequest(context).destination(meetingLocation).origin(departureLocation).mode(TravelMode.WALKING).arrivalTime(endTime);
        DirectionsResult routewalk = bywalk.await();
        DistanceMatrixApiRequest durationWalkRequest = DistanceMatrixApi.newRequest(context).origins(departureLocation).destinations(meetingLocation).mode(TravelMode.WALKING);
        DistanceMatrix matrix = durationWalkRequest.await();
        distances.add(matrix.rows[0].elements[0].duration.inSeconds);
        routes.add(routewalk);
    }

    if (u.travelPref.getPreferences()[3] == true && u.travelPref.getPreferences()[4] == true) {
        DirectionsApiRequest bybusortrain = DirectionsApi.newRequest(context).destination(meetingLocation).origin(departureLocation).mode(TravelMode.TRANSIT).arrivalTime(endTime);
        DirectionsResult routebustrain = bybusortrain.await();
        DistanceMatrixApiRequest durationBusTrainRequest = DistanceMatrixApi.newRequest(context).origins(departureLocation).destinations(meetingLocation).mode(TravelMode.TRANSIT);
        DistanceMatrix matrix = durationBusTrainRequest.await();
        distances.add(matrix.rows[0].elements[0].duration.inSeconds);
        routes.add(routebustrain);
    }
    return routes;
}
```

3.4 addMeeting:

To add a meeting to the database first of all we have to check that there is a free window in the events that are in the database that in our case we will use the Firebase database although for the development with JEE7 we use a different database. In addition to this it is verified that there is no event with that title that must be unique. A meeting is extended from an event. The event has as common variables in breaks and meetings a title, a startTime / startIntervalTime (The second for the break) and an endTime / endIntervalTime. Apart from these variables, a meeting is composed of an ArrayList of routes and two Strings that correspond to the starting point and the location of the meeting. We do not take into account possible errors when the user enters the addresses because we assume that they will always be correct. To get the routes we call the getRoutes function that will be explained later.

Inputs: Title, departure location, meeting location, start time of the meeting and end time of the meeting

Output: Meeting or null.

```
public Meeting addMeeting(String title, String departureLocation,String meetingLocation, ReadWritableInstant startTime, ReadWritableInstant endTime) throws ApiException, InterruptedException{
    if (getMeeting(title) == null && checkTimeSlot(startTime, endTime)) {
        Meeting m = new Meeting();
        m.setDepartureLocation(departureLocation);
        m.setMeetingLocation(meetingLocation);
        m.setTitle(title);
        m.setStartTime(startTime);
        m.setEndTime(endTime);
        ArrayList<DirectionsResult> routes = getRoutes(departureLocation, meetingLocation, endTime);
        m.setRoutes(routes);
        em.persist(m);
        return m;
    } else {
        return null;
    }
}
```

3.5 getStartTimeBreak:

Private function that can get a break start time in the indicated range and with the duration indicated by the user. If all that range is occupied by other events, it will return null. For this we are checking every 30,000 ms (30 seconds) if there is a free time window.

Input: time in milliseconds to rest, start of the interval time that the user want to rest and end of the interval.

Output: start time of the break.

```
private ReadWritableInstant getStartTimeBreak(Long timeRest,ReadWritableInstant startIntervalTime, ReadWritableInstant endIntervalTime){
    ReadWritableInstant startTime = startIntervalTime;
    while (startTime.getMillis() < endIntervalTime.getMillis()){
        ReadWritableInstant endTime = null;
        endTime.setMillis(startTime.getMillis() + timeRest);
        if(checkTimeSlot(startTime,endTime)){
            return startTime;
        } else{

            startTime.setMillis(startTime.getMillis() + 30000);
        }
    }
    return null;

}
```

3.6 addBreak:

To add a break to the database first of all we have to check that there is a free window in the events that are in the database that in our case we will use the Firebase database although for the development with JEE7 we use a different database. In addition to this it is verified that there is no event with that title that must be unique. A break is extended from an event. The event has as common variables in breaks and meetings a title, a startTime / startIntervalTime (The second for the break) and an endTime / endIntervalTime. Apart from these variables, a break is composed of a long variable that corresponds to the time in milliseconds of the rest that is to be carried out in that interval of time specified by the user. To check that there is a free window in our calendar to be able to do the rest check that the private function getStartTimeBreak returns a value other than null.

Inputs: Title, start interval time of the break, end interval time of the break and time to rest.

Output: Break or null.

```
public Break addBreak(String title, Long timeRest , ReadWritableInstant startIntervalTime, ReadWritableInstant endIntervalTime) throws ApiException, InterruptedException{
    if (getBreak(title) == null && getStartTimeBreak(timeRest, startIntervalTime, endIntervalTime) != null) {
        Break b = new Break();
        b.setTitle(title);
        ReadWritableInstant startTime = getStartTimeBreak(timeRest, startIntervalTime, endIntervalTime);
        b.setStartTime(startTime);
        ReadWritableInstant endTime = null;
        endTime.setMillis(startTime.getMillis() + timeRest);
        b.setEndTime(endTime);
        em.persist(b);
        return b;
    } else {
        return null;
    }
}
```

3.7 getWeather:

Function that is responsible for obtaining alerts due to strong wind currents or high probability of precipitation or bad weather. This function will help send notices to the user at the time when the application begins to give directions to reach the destination. For the development of this function we use the YahooWeather API. First, we create a variable of type YahooWeatherService, which is the basis for obtaining the resources of the API. Later we obtain the location of the user and create a channel in that location with temperature measurements in degrees Celsius that will allow us to access the wind speed at that location and the barometric pressure state. The latter

corresponds to an enum with 3 possible values, STEADY, RISING, FALLING. In our case we are interested in the third one that is usually used for the forecast in such a way that if the state in that location at that moment is from FALLING we will give a warning that it is likely to rain so it is not advisable to go cycling or walking. To pass to int the state we use the ordinal() method included in BarometricPressureState. For the wind we check if the speed in km / h is higher than 50km / h, which means strong wind currents. In the same way the user would be notified.

No Inputs.

Output: String with the advise or null if bad weather is not anticipated.

```
private String getWeather() throws JAXBException, IOException{
    YahooWeatherService service = new YahooWeatherService();
    String location = getLocationUser();
    String alert;
    Channel channel = (Channel) service.getForecastForLocation(location, DegreeUnit.CELSIUS);
    Float windspeed = channel.getWind().getSpeed();
    //50Km/h
    if (windspeed > 50){
        alert = "There is to much wind to take the bike or to go walking";
        return alert;
    }
    BarometricPressureState barometricPressure = channel.getAtmosphere().getRising();
    Integer statePressure = barometricPressure.ordinal();
    if (statePressure > 1){
        alert = "It is likely to rain so it is not advisable to use the bicycle or walk";
        return alert;
    }
    return null;
}
```

4.USER INTERFACE DESIGN

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 1: Main Travlendar screen

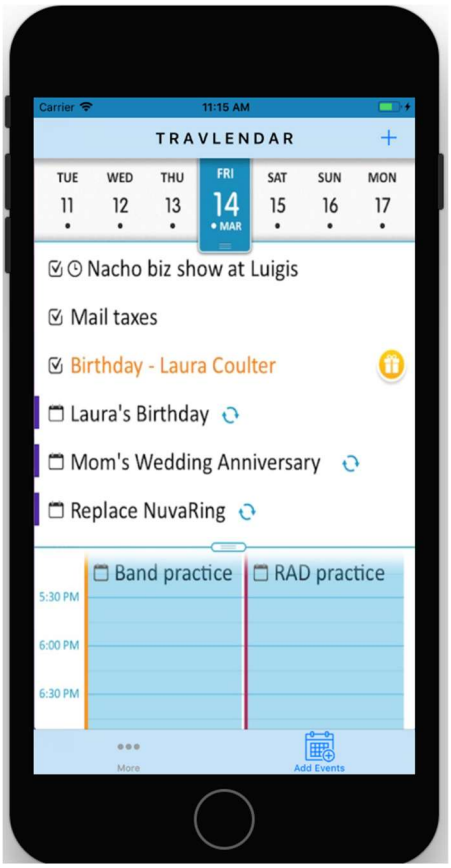


Figure 2: A list of events at a specific date

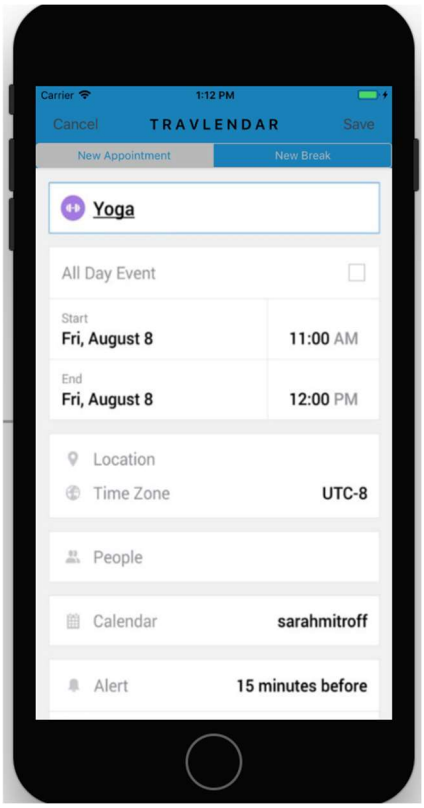


Figure 3: Adding an event

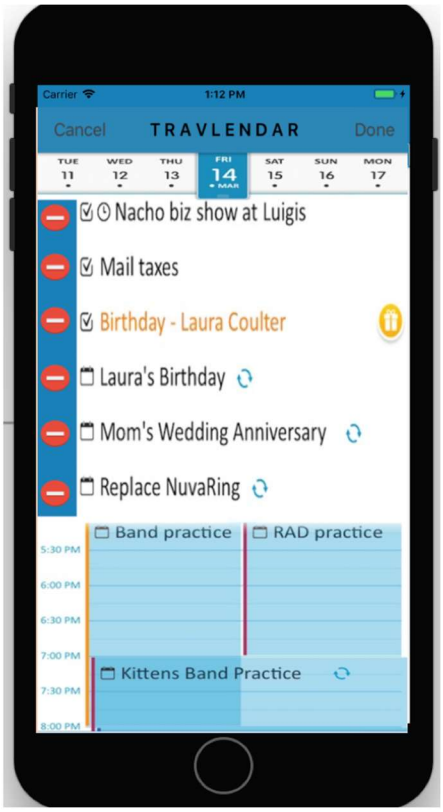


Figure 4: Deleting events

5.REQUIREMENTS TRACEABILITY

- ✓ **[G1] Allow a User to manage appointments:**

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the UserController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the CalendarController component

[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.

- This requirement is implemented in the RouteController component

- ✓ **[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.

- This requirement is implemented in the UserController component

[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.

- This requirement is implemented in the RouteController component

- ✓ **[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.

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the CalendarController component

✓ **[G4] Users should receive alerts**

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

- This requirement is implemented in the CalendarController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the CalendarController component

✓ **[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.

- This requirement is implemented in the UserController component

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

- This requirement is implemented in the RouteController component

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

- This requirement is implemented in the RouteController component

✓ **[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.

- This requirement is implemented in the RouteController component

✓ **[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.

- This requirement is implemented in the RouteController component

✓ **[G8] The system must allow users to log in.**

[R19] The system must allow a user to log in using their Google account.

- This requirement is implemented in the LoginController component

6.IMPLEMENTATION, INTEGRATION AND TEST PLAN

Our aim is to follow the principles defined by Agile software development model. That means that during implementation we can form cross-functional teams and work on different components simultaneously.

The development team will also follow the TDD (Test-Drive-Development) practice which tells us that requirements are first turned into very specific test cases and then software is improved to pass those test cases.

After we create the Router, the plan is to start implementing the LoginController and UserController components first. After that, implementing the RouteController can be possible since it uses UserController for some of its operations. Finally, when we have everything working we can start handling events by implementing the CalendarController.

As mentioned above, unit testing will be done side by side with implementation and new test cases are continuously going to be added while the code evolves.

Acceptance testing will be done in the end of the project with the customer where we will check if all the requirements are met in the final product.

7. EFFORT SPENT

Name	Date	Hours spend
Plamen	12/11/17	1
Plamen	15/11/17	7
Plamen	16/11/17	4
Plamen	17/11/17	1
Plamen	19/11/17	4
Plamen	21/11/17	6
Plamen	22/11/17	2
Plamen	24/11/17	5
Plamen	26/11/17	3
Victor	12/11/17	2
Victor	14/11/17	7
Victor	15/11/17	5
Victor	16/11/17	4
Victor	19/11/17	7
Victor	21/11/17	3
Victor	22/11/17	4
Victor	24/11/17	3

Total: Plamen: 33 hours

Victor: 35 hours

8. REFERENCES

1. Google maps API:
   - https://developers.google.com/maps/web-services/?hl=es-419
   - https://github.com/googlemaps/google-maps-services-java

2. Google Calendar API:
   - https://fullcalendar.io/docs/google_calendar/
   - https://developers.google.com/google-apps/calendar/overview

3. Yahoo Weather API:
   - https://developer.yahoo.com/weather/
   - https://github.com/fedy2/yahoo-weather-java-api

4. Beep slides.

5. Sequence Diagrams:
   - https://www.draw.io

6. User Interface:
   - https://developer.apple.com/xcode/

7. Calendar Notifications:
   - https://developers.google.com/google-apps/calendar/v3/push