Politecnico di Milano
A.A. 2017/2018

TRAVLENDAR+

DD
Design Document

Sara Pidó 894744
Chiara Plizzari 893901
Giuseppe Severino 898458

Contents

1 Introduction

1.1 Purpose

With the Design Document we would like to make the idea of Travlendar+ application more precise and more detailed. In particular the main goal of the DD is to describe the system in terms of architectural design choices. It is written in particular for developers to help them to identify the architectural styles, the design patterns, the main components and their interfaces and, last but not least, the runtime behaviour.

1.2 Scope

The system aims to provide a complete calendar to users which are also helped to find the best route to reach their meetings and their appointments. Users can insert their meetings in order to have an agenda organized in a perfect way: they can see their trips of the day, their itineraries between appointments. Users can choose between different travel options basing on distances, travel time, cost. The system provides some features to personalize the application in order to please users. In fact they can specify lunch time, they can activate or deactivate some travel means, they can also choose for instance to minimize the walking distance or the carbon footprint. With this new application, people and their smartphones can have a lot of appointments in different locations without having the concern about plan the way to reach them. Obviously if two meetings overlap or if it is not possible to reach one, the system will advice users.

1.3 Definitions, Acronyms, Abbreviations

1.4 Definitions

TODO CHECK AND CHANGE –¿ COPIED FROM RASD

- Visitor: a person that is not registered yet, but has the access to the application,Äôs information.
- Registered user: a person that is logged in the system and can create meetings.

- Activity: an event that happens in the real world and that could be a meeting or a break.
- Meeting: an activity among the registered user and other people. It can be created, modified and deleted by the meeting,Äôs creator.
- Break: an activity that a registered user can insert in order to manage it in a customizable way.
- Trip: it indicates the route and the travel means chosen, based on user,Äôs preferences.
- Location: fixed place where a user stands or where he/she has to attend a meeting.
- Global preferences: they are global attributes that registered users can modify and those are valid for all trips (i.e. minimize carbon footprint).
- Creation screen: the screen of the application in which the registered user create a meeting or a break and enters its related details.
- Blocked travel means: it is a travel means that the user has selected as unwanted.
- Warning: a message directed to the user that arrives to him in form of a notification and can be shown on the application screen. It is generated by the system when there are some impediments for a trip (bad weather, strikes, traffic...) or when there are some problems (invalid data during the registration process or during the creation of an activity etc).

1.5 Acronyms

- DD: design document;
- RASD: requirements analysis and specification document;
- API: application programming interface;

1.6 Abbreviations

- Gi: i-goal
- Ri: i-requirement

1.7 Reference Documents

- RASD;
- Specification Document;
- Example of DD of previous years;

1.8 Document Structure

This document is structured as follows:

Section 1: Introduction In this section it is described the purpose and the main goals of the document giving a general description.

Section 2: Architectural Design It gives a general view on how the architecture of Travlendar+ should be showing architectural choices, styles and patterns.

Section 3: Algorithm Design In this part we include the most critical and relevant parts via algorithms.

Section 4: User Interface Design This section provides an overview on how the user will see the application through the mockups and UX and BCE diagrams.

Section 5: Requirements Traceability It explains how requirements defined in the RASD must be mapped to the design elements of the application.

Section 6: Implementation, Integration and Test Plan This part will include the order of implementation of subcomponents and to integrate them. Moreover it will include our plan to test this integration.

Section 7: Effort spent Here are reported the information about the hours of work spent by each member of the group by doing this project.

Section 8: References

2 Architectural Design

2.1 Overview

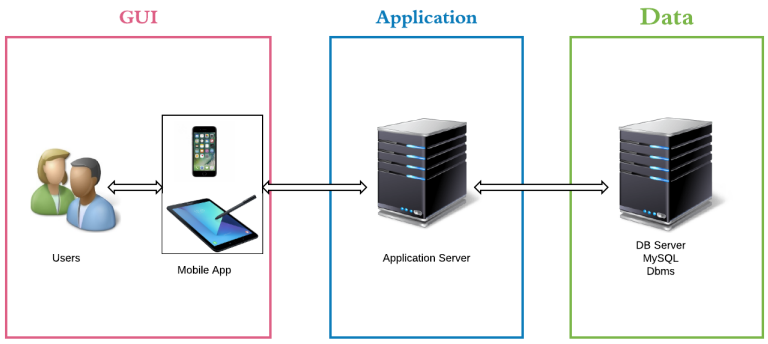
This section deals with the architecture of Travlendar+. It is written to explain architectural choices made for the system. These decisions are supported by different diagrams to highlight the view of components and their interactions and to clarify architectural styles adopted in a better way.

The most suitable system architecture that will allow to satisfy all requirements is a three tier architecture. It is helpful also because allows any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. In this pattern, we have three subsystems to decouple logic and data, logic and presentation:

- Presentation layer: it is the topmost level of the application. It provides a graphic user interface to the client and it communicates with other levels by which it puts out the results to the client displaying information related to the services and also acquires its inputs to send to other tiers.

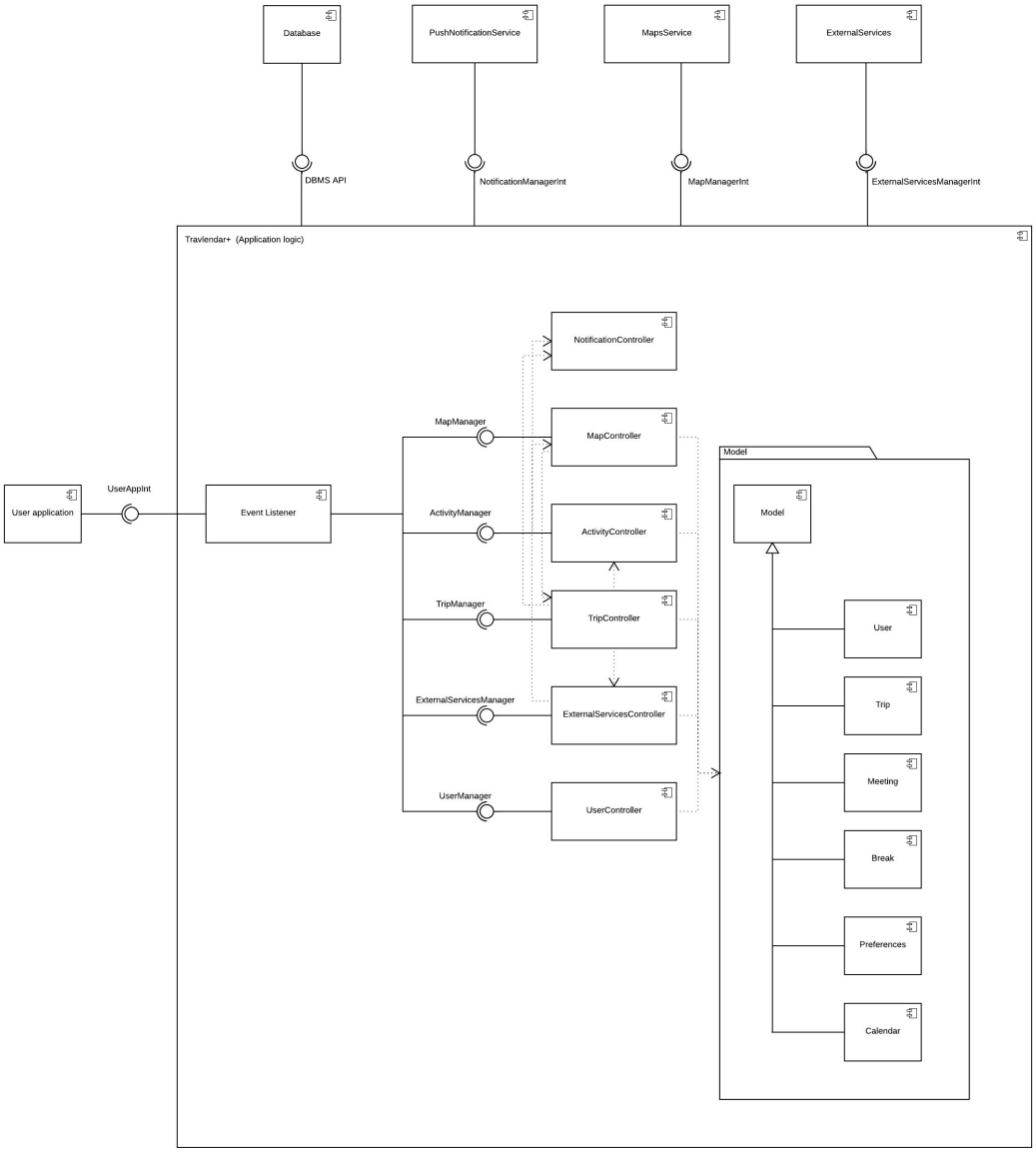
- Application layer: it is the mid level. It controls Travlendar+'s functionalities by performing detailed computation. It coordinates the application, processes commands, makes logical decisions and evaluations and perform calculations. This layer also interacts with external systems that support our application.

- Data layer: it is the level in which information are stored and managed. It includes the data persistence mechanisms (database servers, file shares etc.) and the data access layer that encapsulates the persistence mechanism and exposes the data. It is kept independent from application logic and presentation layer.


GUI
Application
Data
Users
Mobile App
Application Server
DB Server
MySQL
Dbms

2.2 Component View

In this section we will present the whole system. We will focus on all components and their interactions. This diagram will provide also an overview on interfaces that components provide for interactions.



- **Event listener:** it manages the information that it receives from the user application and it sends it to the different components of our application. It decides where the information must be directed. Moreover the event listener forwards the information to the user application.

- **Notification controller:** it manages the warnings, it creates them thanks to the trip controller and sends them to it that will show them to the user.

- **Trip controller:** it occupies of organizing trips between activities. It uses data of the activity controller and it activates the notification controller. It exploits information of the map controller and of the external services controller (such as weather, traffic, strikes).

- **Activity controller:** it receives information from the user to create meetings, modify them and delete them. It receives also data to create, modify and delete breaks. It gives data to the trip controller so that it can organize journeys between them.

- **Map controller:** it gives the opportunity to the user to see information about trips that he/she has to do showing trips on the map. It manages the daily trip between activities through data of the trip controller. ???

- **External services controller:** it manages every external service exploited by our application. It receives information from them and it uses them to provide all functionalities of Travlendar+.

- **User controller:** it manages the information about the user. It receives:

  - personal information such that name, surname, address;
  - data for the login (username and password);
  - global preferences, for example activation or deactivation of travel means or minimum duration of the lunch break.

- **Model:** it is our representation of the world. It contains all data with Travlendar+ deals with.

- **Database:** it is the database to store data persistently.
- **Push notification service:** it manages the sending of all notification to users' devices.
- **Map service:** it provides the map and services like computation of trip distances and travel time
- **External services:** they provides information and functionalities to our application. Travlendar+ exploits data like traffic, strikes, weather and it uses external applications like Ofo, Enjoy, Trenitalia to provide for example the purchase of tickets and the rent of sharing system vehicles.
- **User application:** it is the GUI. It provides the graphic user interface to the client and communicates with the server.

2.2.1 Database

The DB component will include a DBMS to properly manage the data and their handling. It will interact only with the application server. It will use a secure interface so that the date are safely stored and the security is guaranteed. We show the data representation with E-R schema.

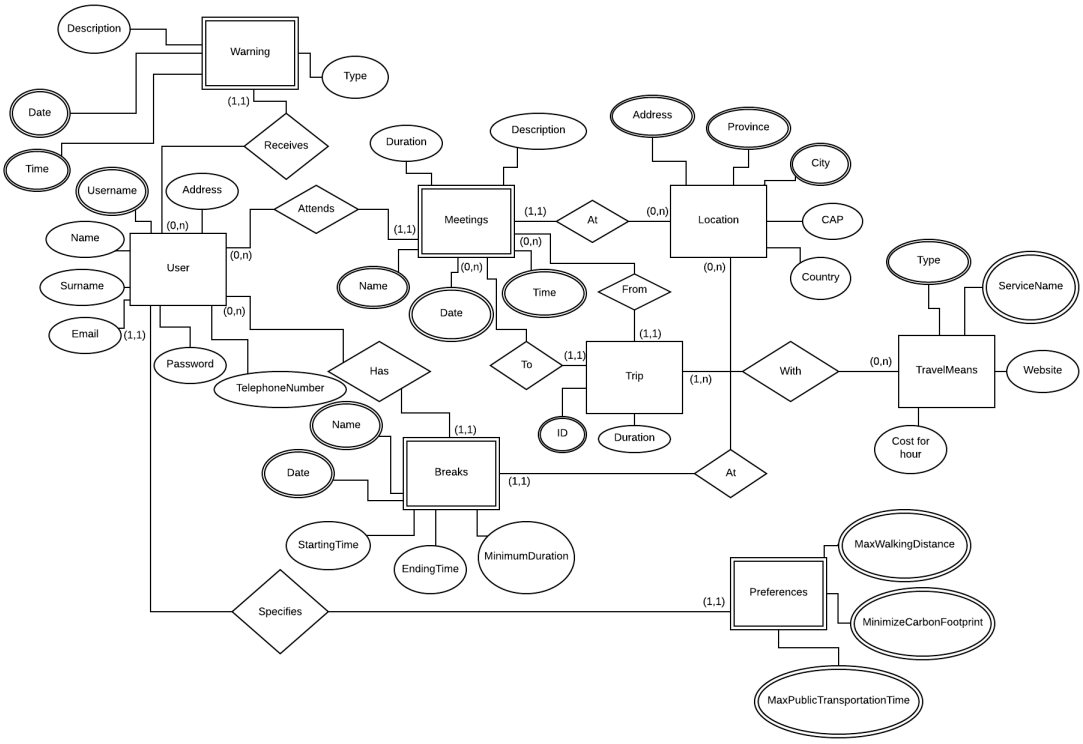


Figure 1: Entity-Relation diagram

2.3 Deployment View

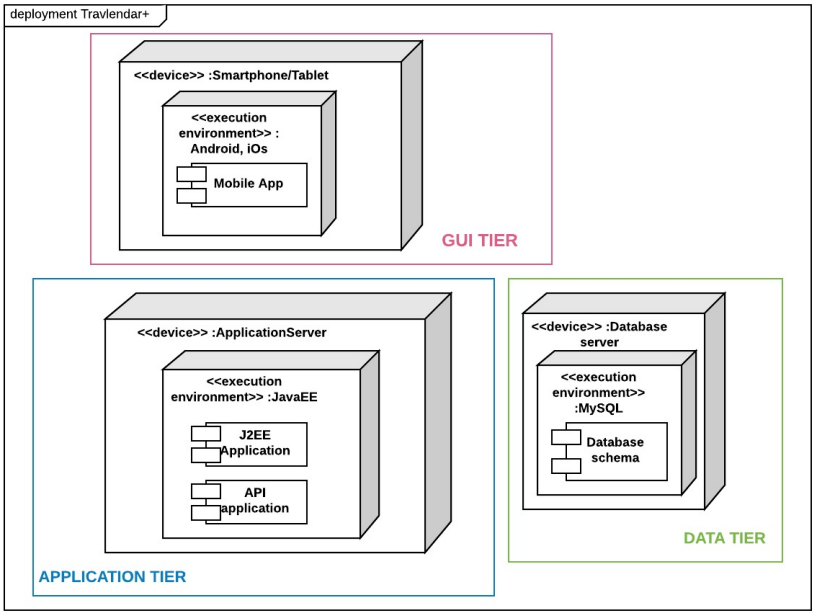


Figure 2: Entity-Relation diagram

In this section we want to show the deployment of our application showing the physical structure of the system. We will use two servers, one for the application and the other for the data. We do not specify the type of the server to use but it should be powerful enough to support all data and all requests that there will be.

The application server will run on the execution environment JavaEE and communication with the database are possible thanks to Java Persistence API which will wrap all database functionalities. The application server will handle all the logic behind the system.

The database server will run on the DBMS with the software MySQL. We will choose this software because it is reliable and moreover it is widely spread because it is free. This server communicates only with the application server.

Clients can use application through smartphones or tablets. The application will communicate with the application server in a direct way.

2.4 Runtime View

We would like to show the behaviour of the application for some of the most important features: through sequence diagrams we explain how the requests of a user are managed by components of Travlendar+.

We think that the behaviour of Travlendar+ is interesting in particular for these functionalities:

- registration;
- login;
- creation of an activity;
- modification of an activity;
- elimination of an activity;
- choice of a trip;
- modification of user's preferences.

Registration

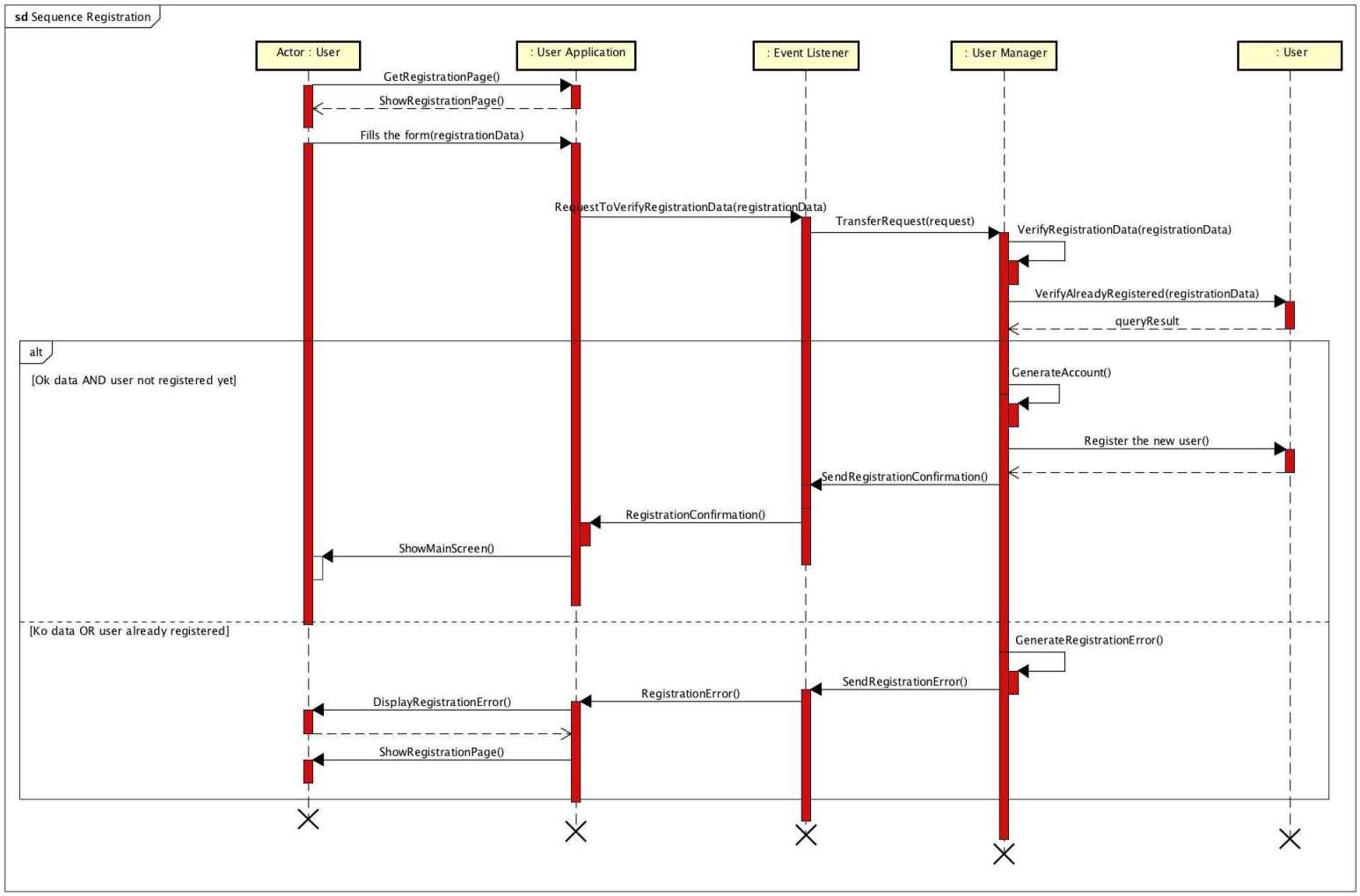


Figure 3: Sequence diagram of the registration process

Login

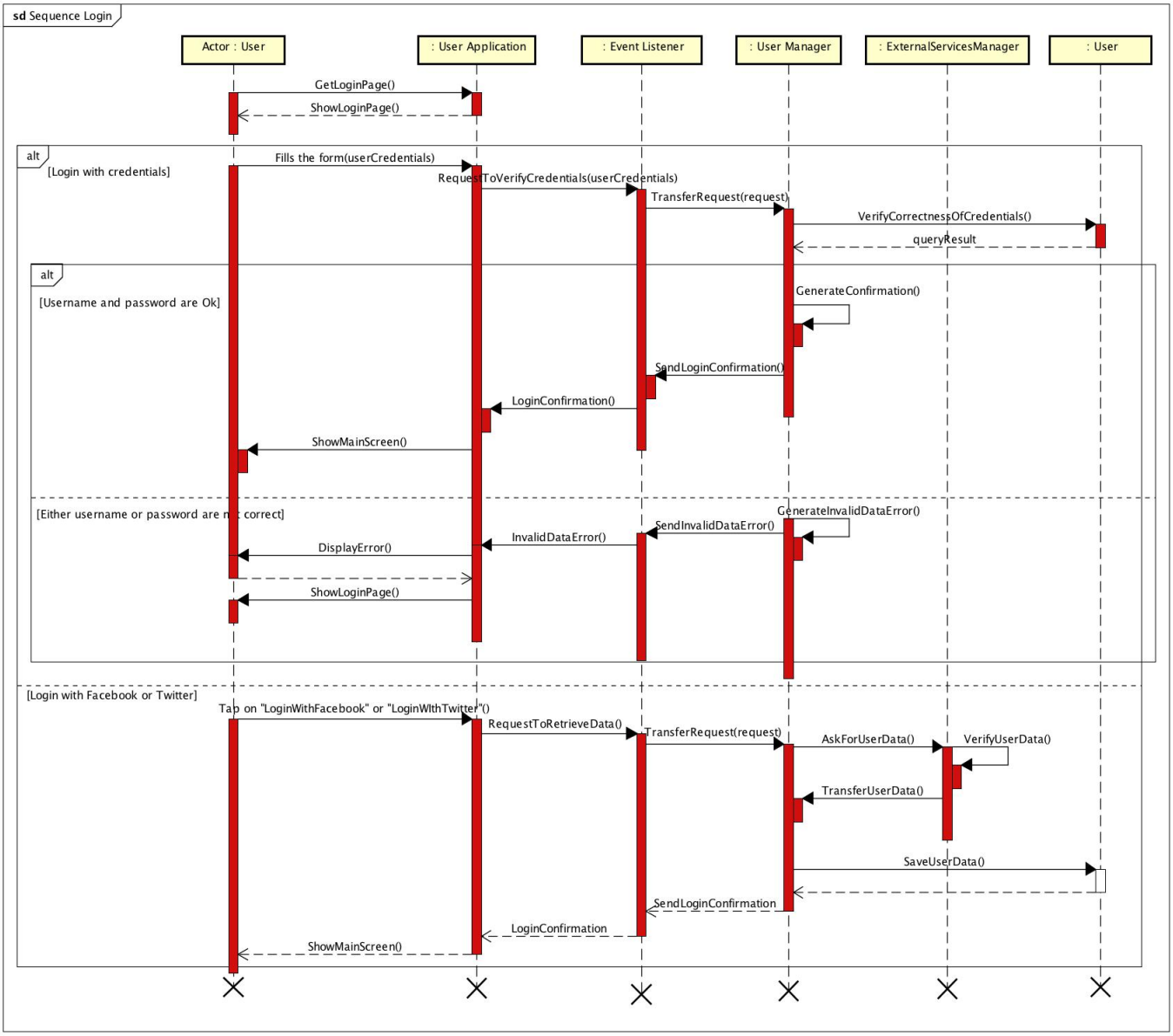


Figure 4: Sequence diagram of the login process

Create an activity

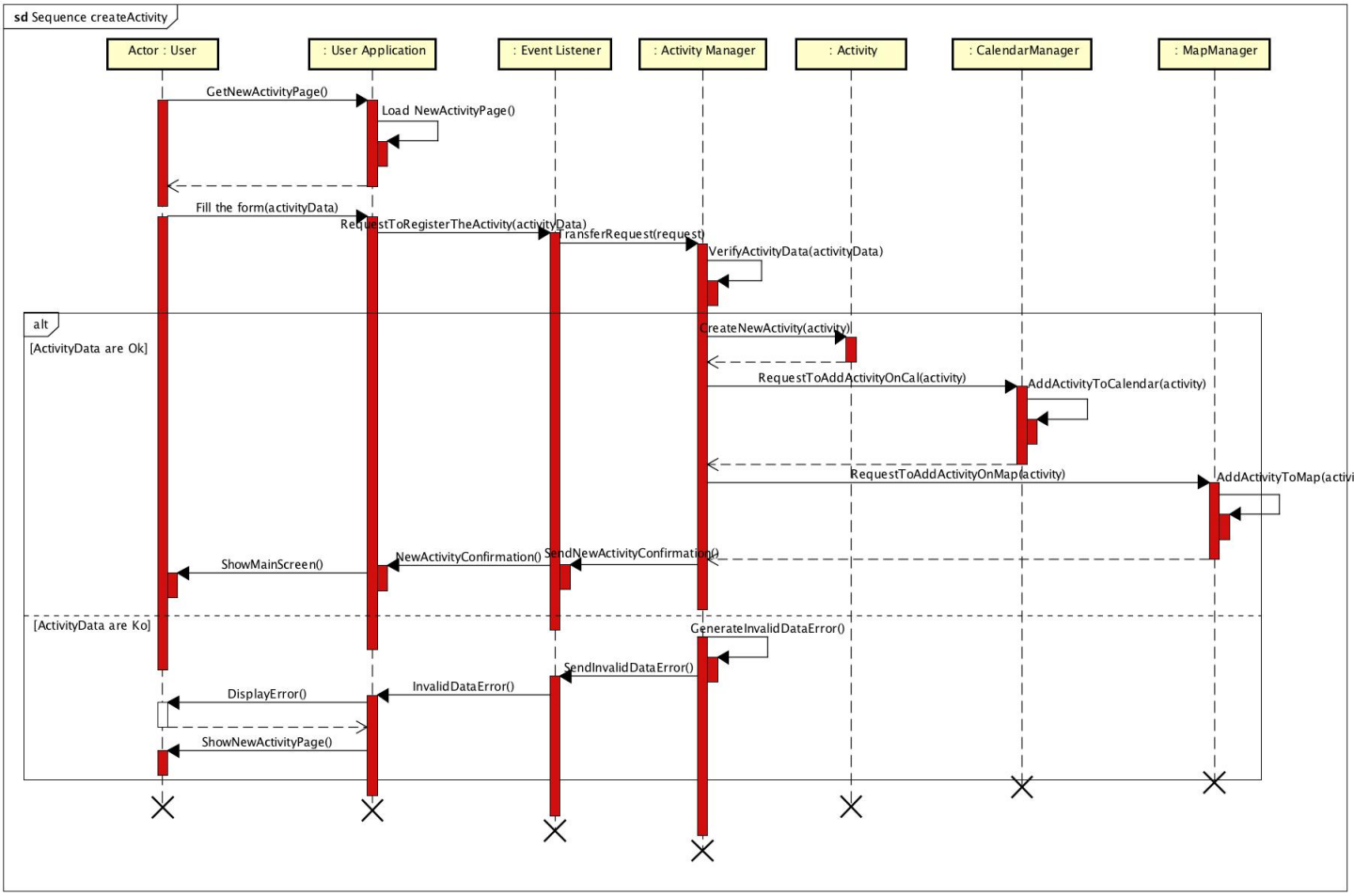


Figure 5: Sequence diagram of the creation of an activity

Modify an activity

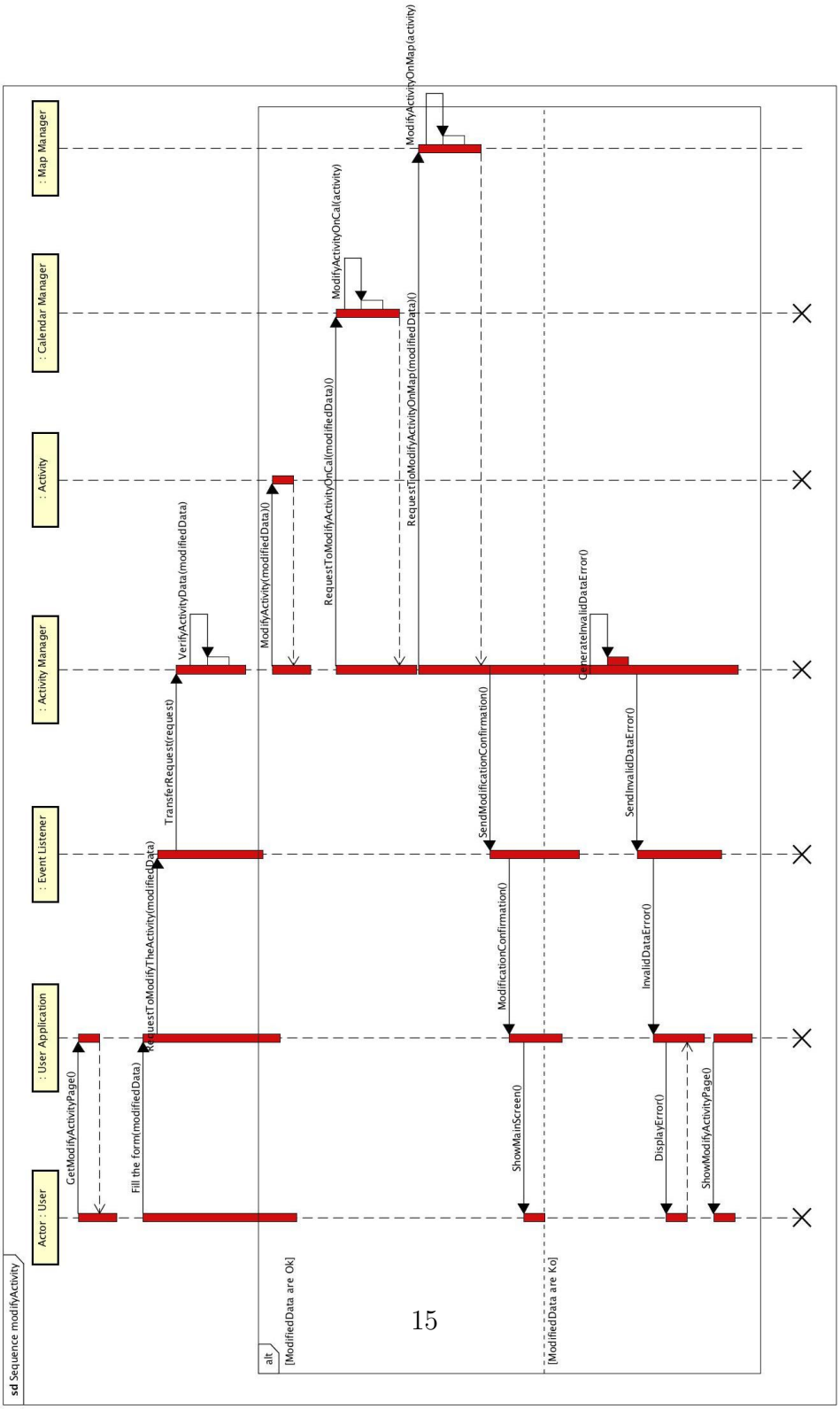


Figure 6: Sequence diagram of the modification of an activity

Delete an activity

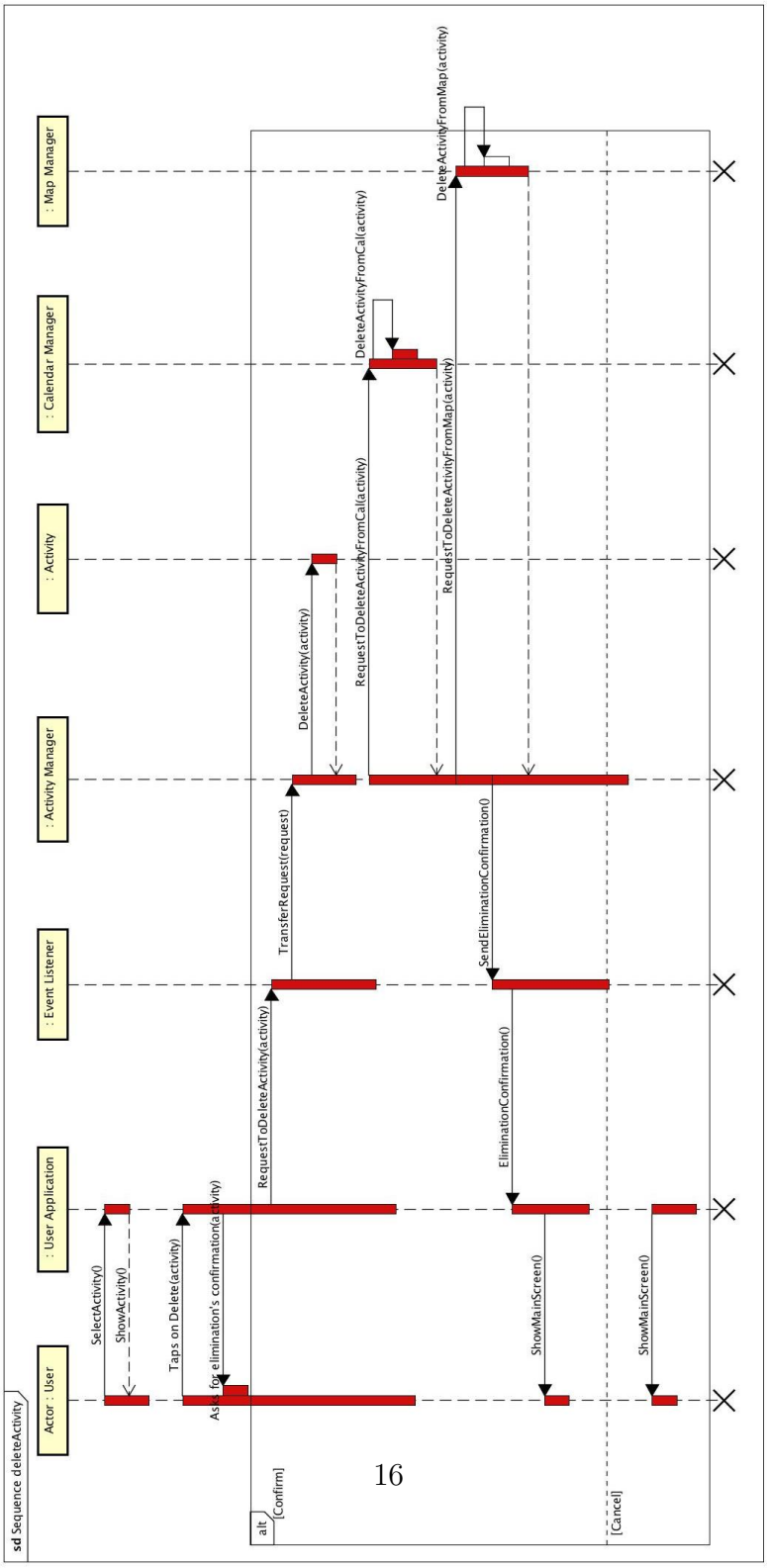


Figure 7: Sequence diagram of elimination of an activity

2.5 Component Interfaces

2.6 Selected Architectural Styles and Patterns

2.6.1 Three-Tier Architecture

As we said before the architectural style that we have chosen for our application is a standard three-tier architecture. The tiers are the following:

1 GUI Tier: Thin clients. They display information and make the different services and functionalities reachable from the client. Clients can communicate with other tiers.

2 Application Tier: It controls application functionality, manages requests coming from the clients and sends results, data and notifications to them. It retrieves data from the Data Tier.

3 Database Tier: Houses database servers where information is stored and retrieved. Data in this tier is kept independent of application servers.

We choose this type of architecture because it has proved to be effective. It allows a developer the opportunity to extend, modularize, and be able to configure their application. Here we list 5 benefits of separating an application into three different tiers:

1 It gives you the ability to update the technology stack of one tier, without impacting other areas of the application.

2 It allows for different development teams to each work on their own areas of expertise.

3 You are able to scale the application up and out, e.g. you can use different technologies to deploy database instead of being stack with only one.

4 It adds reliability and more independence of the underlying servers or services.

5 It provides an ease of maintenance of the code base, managing presentation code and business logic separately.

Moreover with three-tier architecture there is the possibility to use new technologies as soon as they become available. This ensures the product is ready to adapt. There is the possibility to redesign your product.

2.6.2 Design Pattern

Client Server The application is designed to follow the Client-Server communication model. Travlendar+ should be distributed, reachable from a large number of different devices and it needs to provide the service to all of them. We would like to have thin clients: we design the server to manage requests and data, while the client should only see results and provides the user with the possibility of exploiting every functionality that they can use. Moreover the client-server model allows our system to have high maintainability and scalability.

MVC Pattern The application follow the Model-View-Controller software design pattern. This allows our application to be separated into three communicating parts.

- The model represents only the data and nothing else. It does not depend on the controller or on the view.
- The controller provides model data to the view and receives user actions from the view. It depends on the other two parts.
- The view displays the data and sends user actions to the controller.

We choose to use this pattern for different reasons: it fits very well with three-tier architecture, it guarantees more reusability to our application and it provides modularity.

2.7 Other Design Decisions

3 Algorithm Design

In this section we highlight some of the algorithms used for the implementation of the critical part of our application. In particular, the following are the algorithms used for deploying the application in Android (developed using Java language).

3.1 Adding an activity

In this piece of code is shown the function called by the event listener when a registered user insert a new activity (meeting or break) in Travlendar+. It calls the function *checkWarning* from the class *NotificationController* that will be explained in the next subsection.

```
public class ActivityController {
  //Add a new activity to an existing trip
  public void addActivity(RegisteredUser ru, Activity a)
  {
    if (ru.getCalendar().getTrips().getTrip(a.getData()) != null)
        //Create a new trip with the same data of the activity
        ru.calendar().getTrips().createTrip(a.getData());
    //Get the existing trip with the same data of the activity
    Trip currentTrip =
        ru.getCalendar().getTrips().getTrip(a.getData());
    currentTrip.add(a);
    orderActivities(currentTrip);
    //If are errors, delete the activity from the user's trip
    if (NotificationController.checkWarning(t)=true)
      currentTrip.remove(a);
    else
      {
        //Compute and show to the user the path's alternatives to
            reach the meeting
        TripController.computeTripPath(currentTrip);
      }
  }

  //Insertion sort algorithm based on the starting time of
      activities in the trip (converted in minutes for the sake of
      semplicity)
  private void orderActivities(Trip t)
  {
    int i, j;
    for(i=1; i<t.length(); i++)
    {
      int tmp = convertToMinutes(t.get(i).getTime().getHour(),
          t.get(i).getTime().getMinute());
```

```
      for (j = i - 1; (j >= 0) &&
          (convertToMinutes(t.get(j).getTime().getHour(),
          t.get(j).getTime().getMinute()) > tmp); j--)
      {
              t.get(j + 1) = t.get(j);

          }

      t.get(j + 1) = tmp;
    }
  }

  //Utility function that express a time in its equivalent in
      minutes
  private int convertToMinutes(int hour, int minutes)
  {
    return hour*60+minutes;
  }
  ... //Other functions
}


public class TripController {

  public void computeTripPath(RegisteredUser ru, Trip t)
  {
    //Call external services to generate various path alternatives
        (and by using the trip preferences specified by the user)
    ExternalServicesManager.computePath(t);
    //Show the generated alternatives to the user
    ru.show(t.getPathAlternatives());
  }
  ... //Other functions
}
```

3.2 Notification Controller

Below is provided the implementation of the checks that are performed when a new activity is added to an existing trip. When a check returns false, then

a warning is generated and it will appear as pop-up notification in the user's device, by invoking the *sendWarning()* function.

```
public static class NotificationController {

  public static boolean checkWarning(Trip t)
  {
    boolean error = false;
    if (overlappingActivities(t))
    {
      //Create a new warning by specifying (type, description)
      Warning warning = new Warning("OverlappingActivities", "Two
          activities overlap");
      //Activate the function that will send the notification to
          the user's device
      warning.sendWarning();
      error = true;
    }
    if (activityUnreachable(t))
    {
      //Create a new warning by specifying (type, description)
      Warning warning = new Warning("ActivityUnreachable", "The new
          activity make the next activity unreachable");
      //Activate the function that will send the notification to
          the user's device
      warning.sendWarning();
      error = true;
    }
    return error;
  }

  private static boolean overlappingActivities(Trip t)
  {

    for(int i=1; i<t.length(); i++)
    {
      //timeInMinutes1 expresses the activity i-1's starting hour
          in minutes
      int timeInMinutes1 =
          convertToMinutes(t.get(i-1).getTime().getHour(),
```

```
            t.get(i-1).getTime().getMinute());
        //timeInMinutes2 expresses the activity i's starting hour in
            minutes
        int timeInMinutes2 =
            convertToMinutes(t.get(i).getTime().getHour(),
            t[i].getTime().getMinute());
        if (timeInMinutes1+t[i-1].getDuration() > timeInMinutes2)
          return true;

      }
      return false;
    }

    private static boolean activityUnreachable(Trip t)
    {  //t.get(i).getTravelDuration() returns the time for moving
        from the activity i-1 to the activity i.
      //It is already expressed in minutes

      for(int i=1; i<t.length(); i++)
      {
        //timeInMinutes1 expresses the activity i-1's starting hour
            in minutes
        int timeInMinutes1 =
            convertToMinutes(t.get(i-1).getTime().getHour(),
            t[i-1].getTime().getMinute());
        //timeInMinutes2 expresses the activity i's starting hour in
            minutes
        int timeInMinutes2 =
            convertToMinutes(t.get(i).getTime().getHour(),
            t[i].getTime().getMinute());

        if
            (timeInMinutes1+t[i-1].getDuration()+t.get(i).getTravelDuration()
            > timeInMinutes2)
          return true;
      }
      return false;
    }
    ... //Other functions
}
```

4 User Interface Design

5 Requirements Traceability

6 Implementation, Integration and Test Plan

In this part we will define a plan to implement the components of Travlendar+ and to test our application and also the integration between components. We would like to identify:

- objectives and scope;
- documents and items that need to be available to perform the various quality assurance activities;
- items to be tested;
- the analysis and test activities to be performed;

Objectives and scope

Available documents To perform an implementation and an integration test plan with meaningful results, we need that the RASD is available and already completed and discussed at the point the testing will be performed. It describes the application and the functionalities of it. Moreover it is required the DD because it provides an idea of the architecture of the system and it depicts components and their functionalities.

Items to be tested

Analysis and test activites

7 Effort Spent

8 References

For test plan: https://cs.uwaterloo.ca/ gweddell/cs446/ITandPC.pdf http://www.csun.edu/ eugean
4spp.pdf

5.1 Functional Requirements

Requirements of Goals	Components
[$\mathbf{G_1}$] user should be able to go to a meeting.	EventListener ActivityController TripController
[$\mathbf{G_2}$] A user should be able to know if he/she cannot arrive on time to a meeting.	EventListener ActivityController TripController NotificationController
[$\mathbf{G_3}$] A user should be able to have a break that has a certain duration between two meetings.	EventListener ActivityController UserController
[$\mathbf{G_4}$] The time or the location of a meeting can be modified.	EventListener ActivityController MapController
[$\mathbf{G_5}$] A user should be able to unsay a meeting.	EventListener ActivityController
[$\mathbf{G_6}$] A user should be able to have some preferences for the trip.	EventListener UserController TripController
[$\mathbf{G_7}$] A user should be able to know every possible option to go to a meeting with different travel means in order to choose the suitable one.	EventListener TripController MapController
[$\mathbf{G_8}$] A user should be able to buy public transportation tickets or day/week/season pass basing on his/her needs.	EventListener TripController ExternalServicesController
[$\mathbf{G_9}$] A user would like to see on Tripadvisor advised places for his/her breaks.	EventListener ActivityController ExternalServicesController
[$\mathbf{G_{10}}$] A user would like to locate the nearest vehicle of a vehicle sharing system if he/she wants to use that type of travel means.	EventListener ExternalServicesController
[$\mathbf{G_{11}}$] A user would like to have real-time news on strikes, weather and traffic.	EventListener ExternalServicesController