Politecnico di Milano
A.A. 2017-2018
Software Engineering II project

Travlendar+

Design Document

V1

Mirko Mantovani (893784), Matteo Marziali (893904)

November 25, 2017

Contents

1 Introduction

1.1 Purpose

The Design Document is intended to provide a deeper functional description of the Travlendar+ system-to-be by giving technical details and describing the main architectural components as well as their interfaces and their interactions. The relations among the different modules are pointed out using UML standards and other useful diagrams showing the structure of the system. The document aims to guide the software development team to implement the architecture of the project, by providing a stable reference and a unified vision over all parts of the software itself and clearly defining how every part interacts with the others

In summary the document will help the developers to create the software by referring to:

- A high level architecture.
- The design patterns to use.
- The main components and the interfaces they provide to communicate with one another.
- The Runtime behaviour of the system.

1.2 Scope

The main focus of our system design phase will be to create an application capable of reaching the vast majority of users. Thus the architecture must be designed with the intent of being maintainable and extensible, also forseeing future changes. It should also be flexible enough in order to make future integration of features or adaptations and deploy on other type of platforms and devices as easy as possible. This document aims to drive the implementation phase so that cohesion and decoupling are increased in full measure. In order to do so, individual components must not include too many unrelated functionalities and they should reduce interdependency between one another.

1.3 Definition and Acronyms

1.3.1 Definitions

- **App:** this is the abbreviation for application, in particular this term is used meaning a mobile application.
- **Delay notification function**: this phrase refers to the function which allows to notify the participants of a meeting through an email in case the user is late.
- **Travel:** a travel is any suggested path that goes from the starting point to the meeting location.
- **Route:** this term is used as a synonym of travel.
- **Warning:** warning is the word used to define the conflict between two meetings.
- **Conflict:** a conflict between two or more meetings is what enables the creation of a warning, it means that the set of meetings in conflict are scheduled too close in time in order for the user to be able to attend them all in time.
- **Calendar:** the calendar contains the list of meetings and is grouped by day.
- **Meeting:** is an important keyword of the application, it includes all the informations of an appointment.
- **Reminder:** a reminder is a sort of an alarm triggered at a certain time before an appointment is starting.

1.3.2 Acronyms

- **API**: Application Programming Interface
- **JEE**: Java Enterprise Edition
- **EJB**: Enterprise Java Bean
- **JPA**: Java Persistence API
- **JSP**: Java Server Pages

1.4 Revision

1.5 References

- The document with the assignment for the project
- The RASD document of Travlendar+

1.6 Document Structure

This document is structured in seven sections, here is an overview of the contents of each and every one:

- **Introduction**: This section provides a general introduction and overview of the Design Document and the covered topics that were not previously taken into account by the RASD.
- **Architectural Design**: This section shows the main system components together with sub-components and their relationship. This section is divided into different parts whose focus is mainly on design choices, interactions, used architectural styles and patterns.
- **Algorithm Design**: This section provides a high-level description and details about some of the most crucial and critical algorithms to be implemented in the system-to-be.
- **User Interface Design**: It provides an overview on how the user interface will look like and behave giving a more complete view with respect to those contained in the RASD.
- **Requirements traceability**: This section describes how the requirements defined in the RASD are mapped and are satisfied by the design elements and components defined in this document.
- **Implementation, integration and test plan**: This section is used to explain the strategies for implementations and testing that will be adopted in the development part of the project.
- **Appendix**: Here we provide information about the used software and the effort spent to redact this document.

2 Architectural Design

2.1 Overview

This section of the document gives a detailed view of the physical and logical infrastructure of the system-to-be.
It provides the different types of view over the sysyem as well as the description of the main components and their interactions.
A top down approach will be adopted for the description of the architectural design of the system:

Section 2.2 High-Level components: A description of high-level components and their interactions.

Section 2.3 Component View: A detailed insight of the components described in the previous section.

Section 2.4 Deployment view: A set of indications on how to deploy the illustrated components on physical tiers.

Section 2.5 Runtime View: A thorough description of the dynamic behaviour of the software with diagrams for the key-functionalities.

Section 2.6 Component Interfaces: A description of the different types of interfaces among the various described components.

Section 2.7 Selected Architectural styles and patterns: A list of the architectural styles, design patterns and paradigms adopted in the design phase.

Section 2.8 Other design decisions: A list of all other relevant design decisions that were not mentioned before.

2.2 High-Level components: general architecture identification

The design approach is a JEE Architecture which is based on a client-server 4-tier distributed system.
Here we provide for each tier the definition, choice reasons and used technology:

- **Client Tier:** this tier is responsible of translating user actions and presenting the output of tasks and results into something the user can understand;
- **Web Tier:** it receives the requests from the client tier and forwards the pieces of data collected to the business tier waiting for processed data to be sent to the client tier.
  Web Tier is composed by web beans. This tier purpose is the one to interact with the beans in the Business Logic tier and display data according to the user requests.
- **Business Logic Tier:** this tier contains the business logic, it coordinates the application, processes commands, makes logical decisions and evaluations and performs computations.
  It is responsible for the communication between the Web Tier and the Persistence Tier. Its components are the EJB Beans.
- **Persistence Tier:** this tier holds the information of the system data model and is in charge of storing and retrieving information from the database.
  The persistence tier is composed of the entity beans which represent the entities depicted from our RASD document and then further endorsed in our conceptual design. These entities are fundamental as they represent the connection to our database. Since in JEE we are interested in working in an object oriented environment, they represent a high level object view of the database.
  In particular, for Travlendar+ it will be used a relational DBMS: MySQL, and the JPA standards of JEE in order to look and use the database entities in a object oriented way.

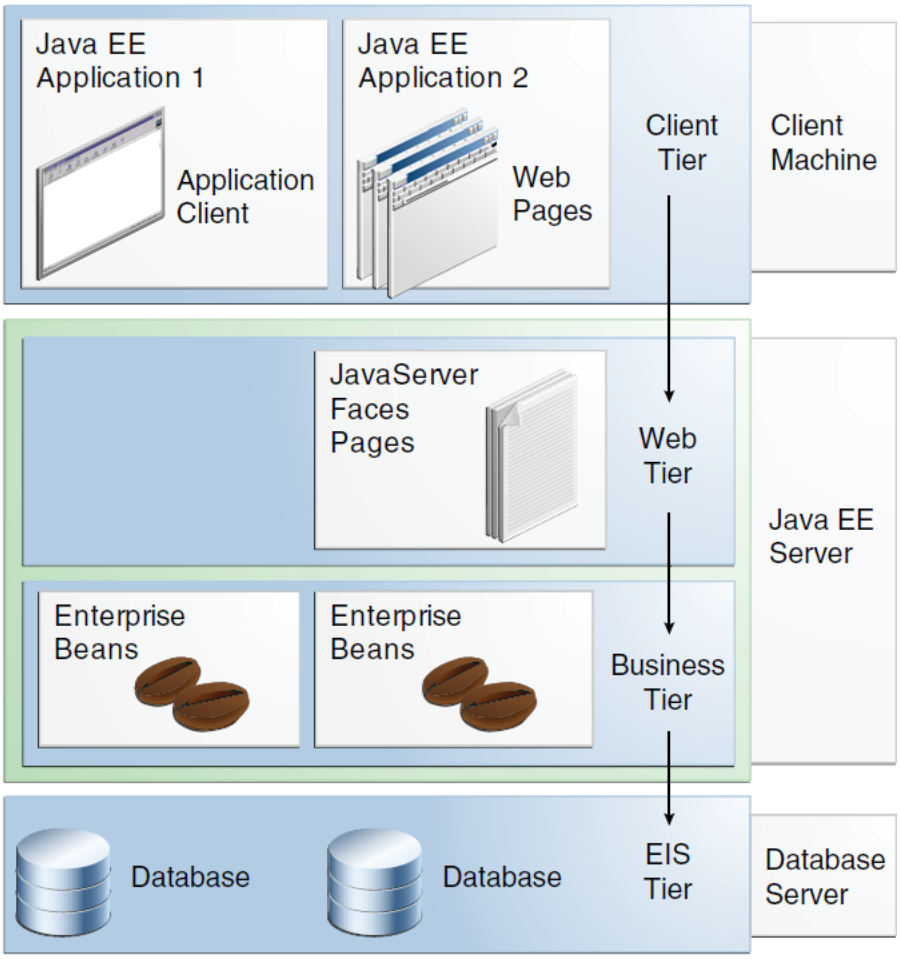

Figure 1: JEE architecture

2.3 Component View

2.3.1 Database

The persistence layer must include a DBMS component, in order to manage the insertion, modication, deletion of data and managing the relative transactions on the database.
Regardless of the implementation, the DBMS must guarantee the correct functioning of concurrent transactions and the ACID properties; it also must be a relational DBMS, since the application needs in terms of data storage do not require a more complex structure than the simple one provided by the relational data structure.
The data layer must only be accessible through the Application Server via a dedicated interface. With respect to this, the Application Server must provide a persistence unit to handle the dynamic behaviour of all of the persistent application data.
Sensible data such as passwords and personal information must be encrypted properly before being stored. Users must be granted access only upon provision of correct and valid credentials.
The E-R diagram illustrates a detailed view over the database schemas and attributes.

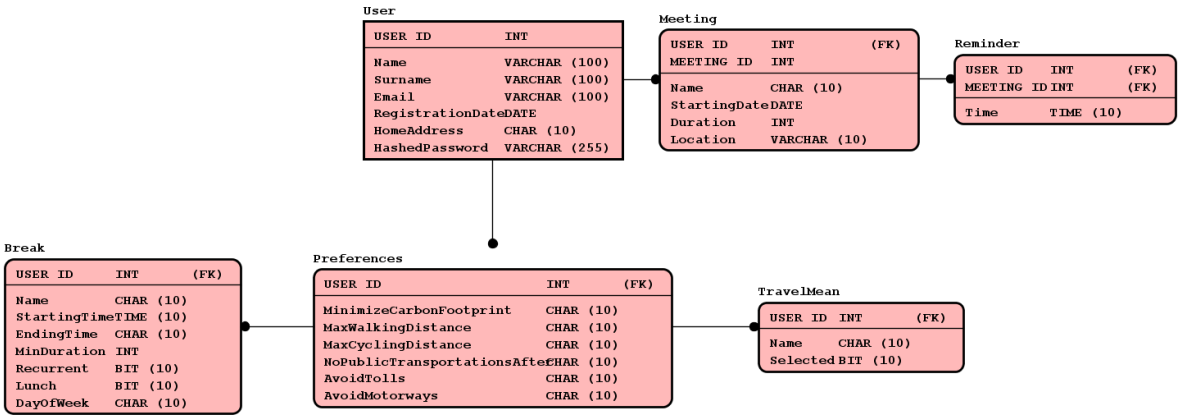

Figure 2: The E-R diagram of the database schema.

We also provide a projection of the E-R diagram in the form of a class diagram:

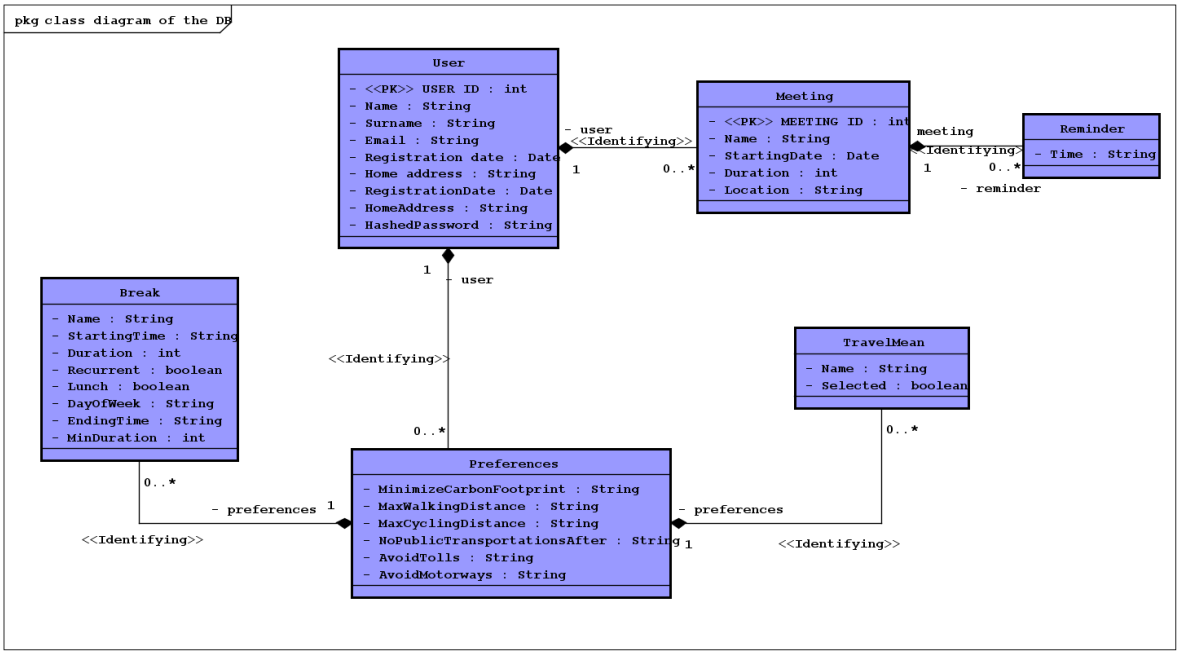

Figure 3: The E-R diagram of the database schema.

2.4 Deployment View

2.5 Runtime View

2.6 Component Interfaces

2.7 Selected architectural styles and patterns

2.7.1 MVC Pattern

Model-View-Controller pattern divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. This is one of the most common and effective ways to avoid a dangerous level of coupling between the various parts of the whole system.

2.7.2 4-tier Architecture

A standard JEE 4-tier architecture was used to design the system, where the layers are, as better described in [Section 2.2] **High-Level components**:

- **Client Tier**
- **Web Tier**
- **Business Logic Tier**
- **Persistence Tier**

2.7.3 Client-Server

A client-server programming approach was used to design the application, in particular, the choice was to create a thin client, and makes all the logic and controlling reside in the Application server, which must have enough computing power to manage concurrent accesses in an efficient way. On the other hand, the mobile application is only in charge of the presentation and it does not involve decision logic.

2.7.4 Deployment

- **Divide et impera:** software is divided in 4 independent tiers that put together offer the Travlendar+ application services.
- **Cohesion:** each module has his specific purpose; Client tier handles the system clients; Logic tier handles the application processing and the Data layer that handles and manages the persistence of data.
- **Decoupling:** the sofware components are independent and communicates through interfaces
- **Flexibility:** implied from decoupling, abstraction and reusability properties.
- **Anticipating obsolescence:** if the technologies that compose a tier become obsolete, a tier can be substituted without affecting the others.
- **Portability:** each tier can be run on different platforms.

2.7.5 Structure style

An Object-Oriented Architectural Style was chosen. It supports the division of responsibilities for a complex system into small and reusable parts called "Objects".
They communicate with each other through interfaces, by sending and receiving messages or by calling methods on other objects. The main benefits are:

- **Extensibility:** change of implementation does not imply an interface change
- **Reusability:** objects are developed as small reusable piece of software
- **Testability:** encapsulation improves testability

2.8 Other design decisions

3 Algorithm Design

3.1 Meeting Warning checker algorithm

One of the fundamental identified algorithms is the one that checks whether a newly added meeting can generate a warning with the previous meeting, the following, or both of them. If it's one of those cases a warning containing the correct meetings is created in the system (see createWarning method in the code).
The pseudocode of the algorithm is provided below:

```
BEGIN checkWarningPresence(Meeting newMeeting)
//suppose we can get the entire list of Meetings in an array
//ordered by starting date

Meeting meetings[ ]:= READ from DB
int index:= variable that is going to assume the value of
the position of the array in which newMeeting should be
put in order for the array of meetings to remain ordered
by starting date

index=binarySearch(meetings,newMeeting)


Meeting prevMeeting=meetings[index-1]
Meeting nextMeeting=meetings[index+1]

//Here external API's will be used in order to compute
//the minimum travel time
Time prevTime=computeMinTravelTime(prevMeeting, newMeeting)
Time nextTime=computeMinTravelTime(prevMeeting, newMeeting)

//case 1: conflict with both the previous and next meeting
IF newMeeting.startTime-prevMeeting.startTime+
prevMeeting.duration < prevTime AND
 nextMeeting.startTime-newMeeting.startTime+
prevMeeting.duration < nextTime
THEN
  createWarning(prevMeeting,newMeeting, nextMeeting)

ELSE
    //case 2: conflict only with the previous meeting
    IF newMeeting.startTime-prevMeeting.startTime+
    prevMeeting.duration < prevTime
    THEN
      createWarning(prevMeeting,newMeeting)

    ELSE
  //case 3: conflict only with the following meeting
```

```
        IF nextMeeting.startTime-newMeeting.startTime+
        prevMeeting.duration < nextTime
        THEN
            createWarning(newMeeting, nextMeeting)

        END-IF
    END-IF
END-IF

END
```

3.2 Break Warning checked algorithm

Another important algorithm is the one that checks whether a newly added meeting can generate a warning with one or more breaks, this is a little bit more complicated because in the most general case no one forbids a user to insert overlappings breaks, more meetings entirely within the limits of a single break or even nested breaks.
The proposed algorithm will be able to handle all the various cases. What has not yet taken into consideration is that if a meeting is in conflict with a break, this does not necessarily mean that the only way to solve this conflict is to modify THIS precise meeting that was discovered as problematic, another possibility could be to shift other meetings happening in the break limits range in order to make space for the minimum duration of the break to fit in.
The pseudocode of the algorithm is provided below:

```
BEGIN checkBreakConflict(Meeting newMeeting)

//the array breaks will contain all the breaks which
//overlaps with the new meeting which is being added
Break breaks[ ]:= READ from DB

//A break has a startingTime, and endingTime, and a
//minDuration. This allows a break to actually happen
//within the limits given by startingTime and endingTime
//and to have a minumum duration of minDuration, the
//breaks can shift in the allowed range of time

FOR i=0 TO breaks.length-1 DO

  //overlappingMeetings array will contain all and
  //only the meetings whose schedule is overlapping
  //with the i-th break, this means that either the
  //meeting startTime is betweeen the break
  //startingTime and endingTime or the meeting
  //startTime + its duration is.
  Meeting overlappingMeetings[ ]:=
```

```
22        getOverlappingMeetings(breaks[i])
23
24    //The array will be ordered by start date
25    orderByStartDate(overlappingMeetings)
26
27    //the new meeting is added in the list respecting
28    //the established order
29    insertOrdered(overlappingMeetings,newMeeting)
30
31    //An ad hoc class is used in order to better explain
32    //a possible algorithm to solve the problem.
33    //Interval will only have start and end fields.
34    Interval remainingSlots[ ]:= array of free intervals
35          within the breaks limits
36
37    IF overlappingMeetings[0].startTime > break[i].
        startingTime THEN
38        remainingSlots.add(break[i].startingTime,
        overlappingMeetings[0].startTime)
39    END-IF
40
41
42    FOR j=0 TO overlappingMeetings.length-2 DO
43        remainingSlots.add(overlappingMeetings[j].endTime,
44        overlappingMeetings[j+1].startTime)
45    END-FOR
46
47    IF overlappingMeetings[overlappingMeetings.length-1].
        endTime < break[i].endingTime THEN
48        remainingSlots.add(overlappingMeetings[
        overlappingMeetings.length-1].endTime,
49         break[i].endingTime)
50    END-IF
51
52    //Declaring a flag that states the feasibility of the
        break
53    Boolean breakIsPossible:= false
54
55    FOR k=0 TO remainingSlots.length DO
56        IF remainingSlots[k].end-remainingSlots[k].start
57          >= break[i].minDuration THEN
58            breakIsPossible=true
59        END-IF
60    END-FOR
61
62    IF !breakIsPossible THEN
63        createBreakWarning(break[i],newMeeting)
64
65
66 END-FOR
```

```
67
68 END
```

4 User Interface Design

4.1 UX Diagram

As a way to show the navigation among the different pages and define the visual flow of screens we redesigned a completed User Experience diagram starting from the draft that was introduced in the RASD document.

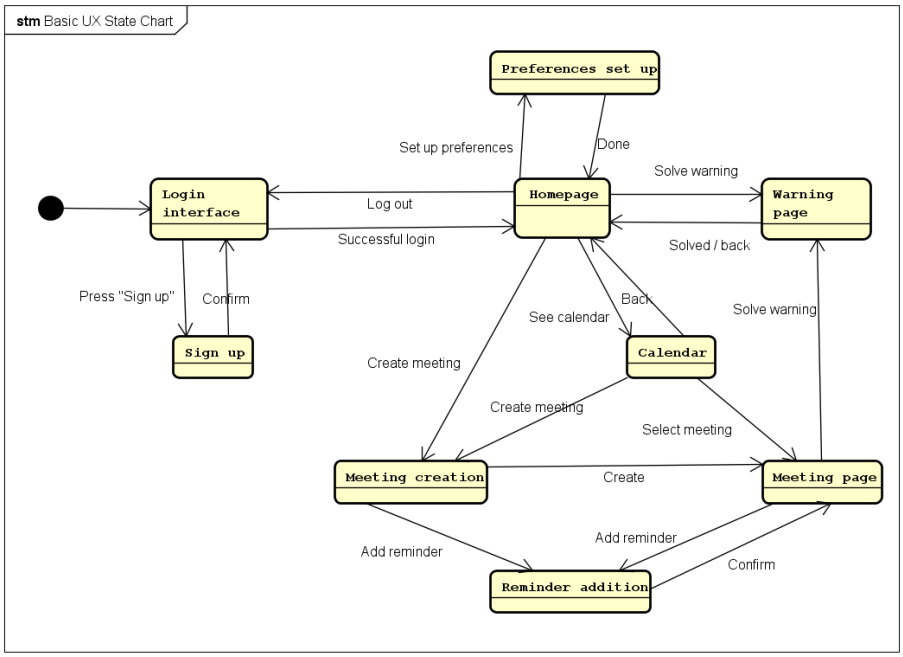

Figure 4: The initial UX diagram

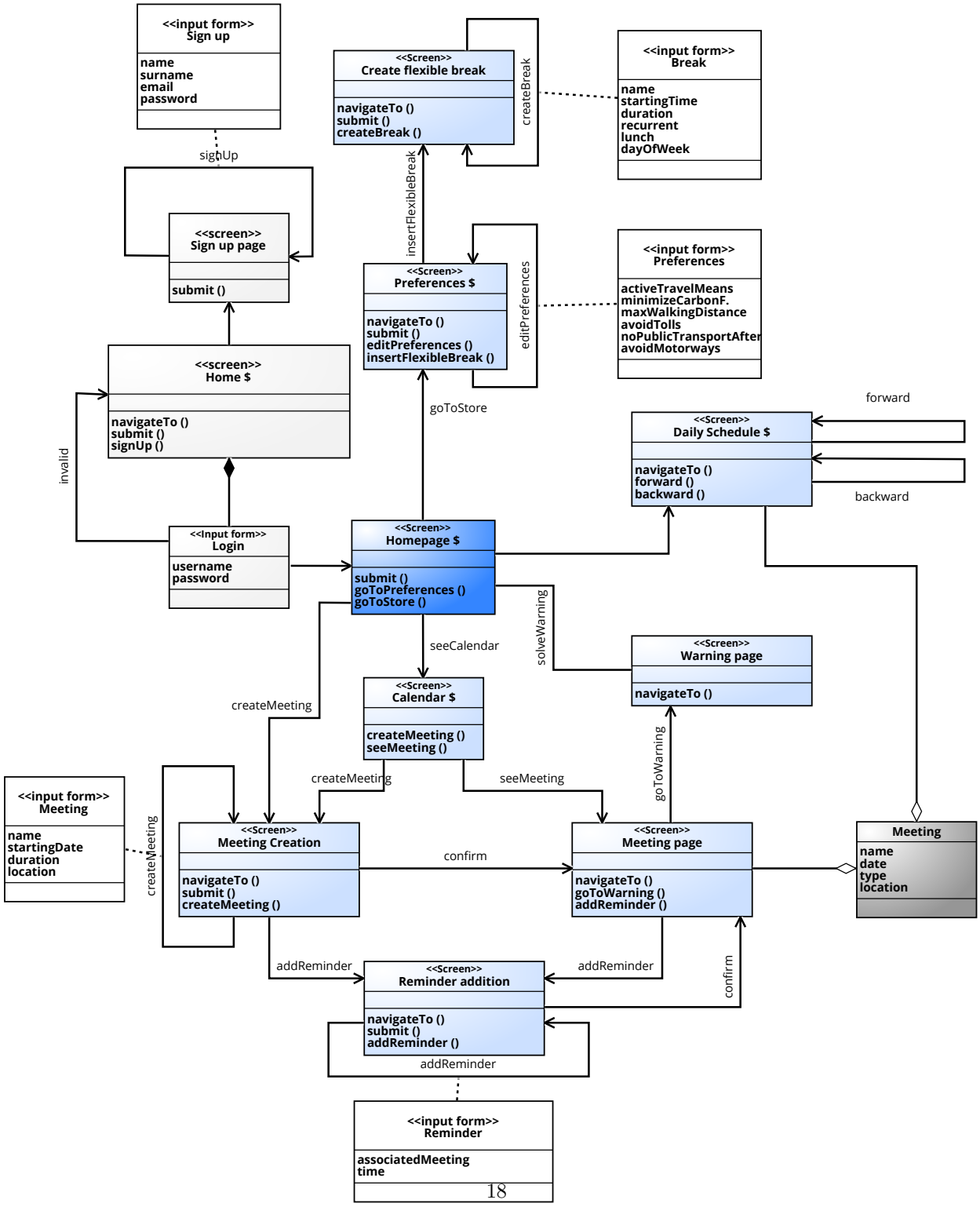

Figure 5: The complete User Experience diagram of the application

4.2 Mockups of the User Interface

Concerning the user interface requirements, we established to directly provide information about the application screens and layout through several mockups. We designed the sketch of the main pages of Travlendar+ following our aim to make a light and user friendly product but, at the same time, we have pursued an attractive and impressive style.
Apparently, the coding phase could affect our desing. Hence, these sketches are not definitive, their aim is to give an idea of the application design.
For this reason, if we notice that some changes are necessary due to either app. improvement or obstacles in realization, we would be ready to modify them.

4.2.1 Homepage

To pursue our aim of realizing a user-friendly and light app, we decided to provide the main functions directly in the homepage. Indeed, from this page, a guest can reach the sign up form, a user can log himself in,can insert a meeting by tapping onto a day in the calendar,can manage his preferences and even access the warnings.

Figure 6: Travlendar+ homepage.

4.2.2 Quick Menu

We thought about a quick menu to collect some secondary functions, to make them easily reachable. By tapping on the top-left corner of the app people have the opportunity to either register or log in themselves, to only view meetings of the current day, to access the reminders they set up, to manage the warnings and to logout if they are already logged in.

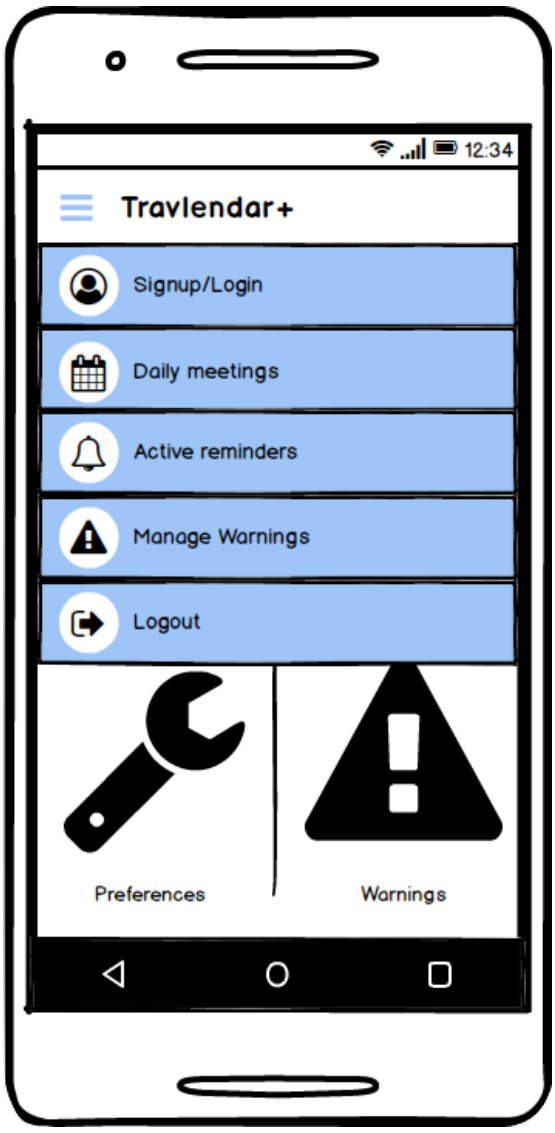

Figure 7: Quick access menu.

4.2.3 Meeting Creation

This page is a very simple form, which allows the user to finalize the creation of a meeting by filling it in all its fields. The house-logo in the right corner,on the top band and near the application name, represents the return-to-homepage icon.

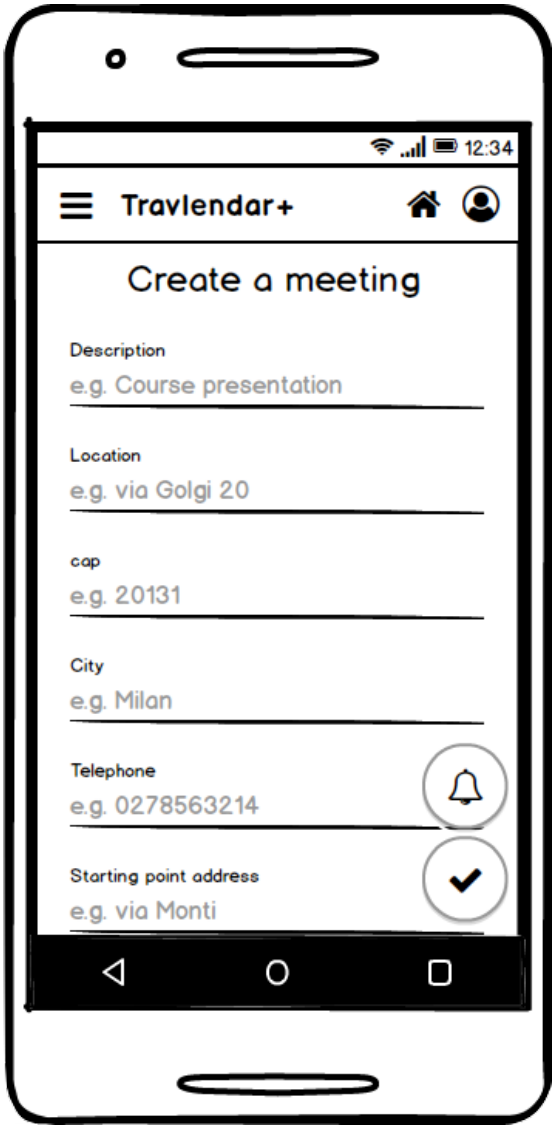

Figure 8: Meeting creation page.

4.2.4 Meeting Page

This screen wants to provide all the useful information related to a meeting already registered in the system. First details provided, in the highest portion of the page, regard the meeting location and the route to reach the appointment, further information are located below. In addition, on the right part of the screen, there are quick access buttons: the "plus" icon allows the user to add a reminder for the event, the "pencil" icon is to change meeting details and the "x" button provides deletion function.

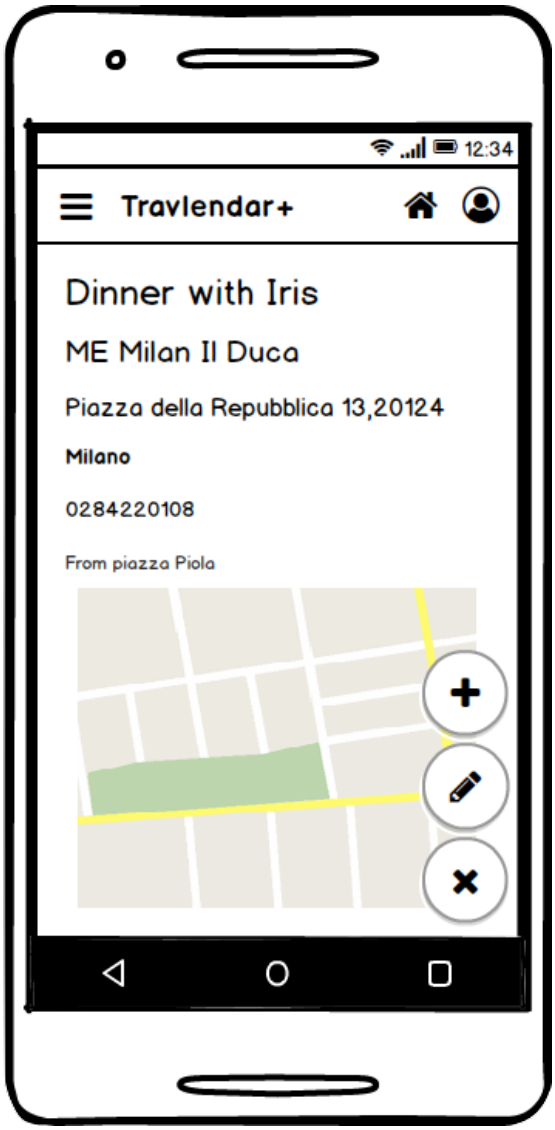

Figure 9: Meeting view page.

4.2.5 Warning Page

The warnings page has the role to summarize and notify all the conflicts among meetings which involve the user's appointments. Every warning is represented by a dialog which points out the meetings that generate the conflict and which has two buttons, one to ignore the warning and other one to solve it by modifying the conflictual meetings. To prevent too much user's clicks, an "ignore all" button is provided, which is equivalent to tap "ignore" for each warning in the list.

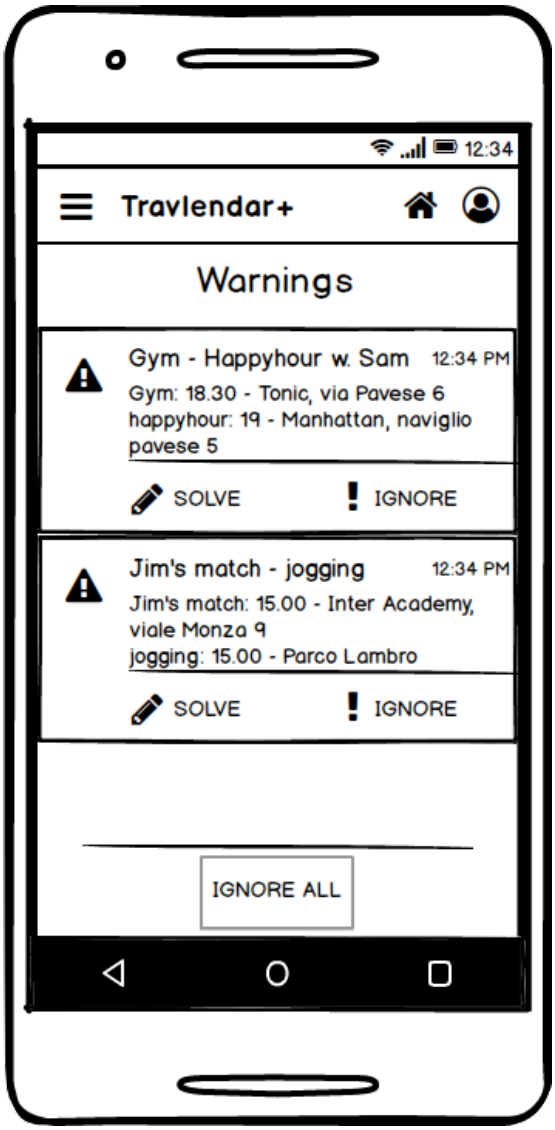

Figure 10: Warnings page

4.2.6 Travel means preferences

This is a very simple page with a minimalistic design, not to uselessly load the application. However, this basical view provides all the functions which the user needs to select his preferences related to the transports for his travels. Please notice that the preferences section can be changed by tapping on an specific topic just below the "Preferences" bar.

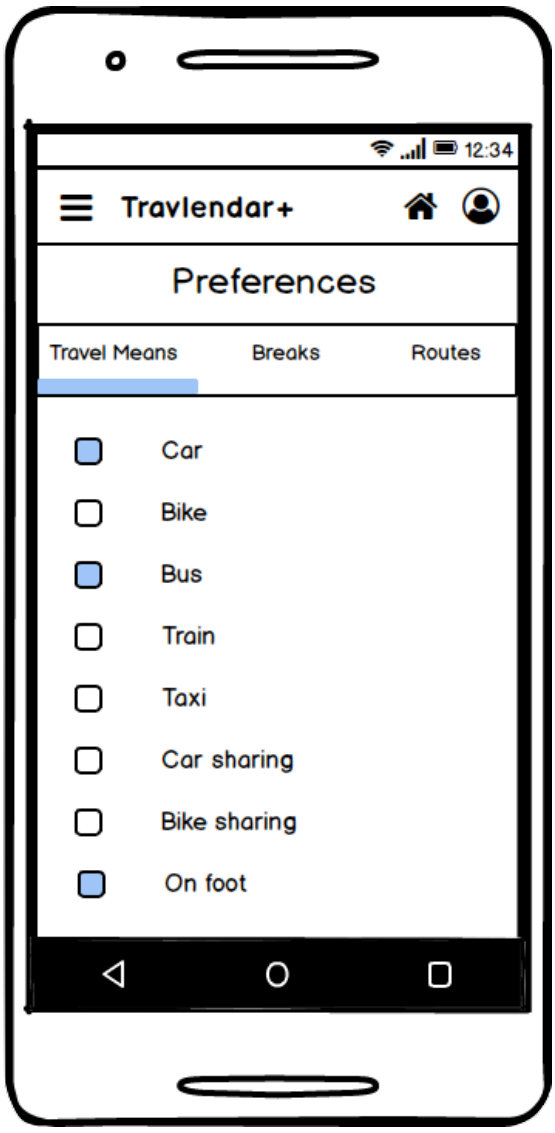

Figure 11: Travel mean preferences page.

4.2.7 Breaks preferences

The breaks page consists in a list of events, that represent pauses, organized in order of creation and labeled by the type of the break. For every break both the starting and the ending time, a "pencil" button to allow modifications and a "x" button,to remove it, are provided. In addition, thanks to the fact that we decided to make the breaks general, that means not related to a specific day, the user has the faculty to either flag or unflag them to activate or deactivate them.

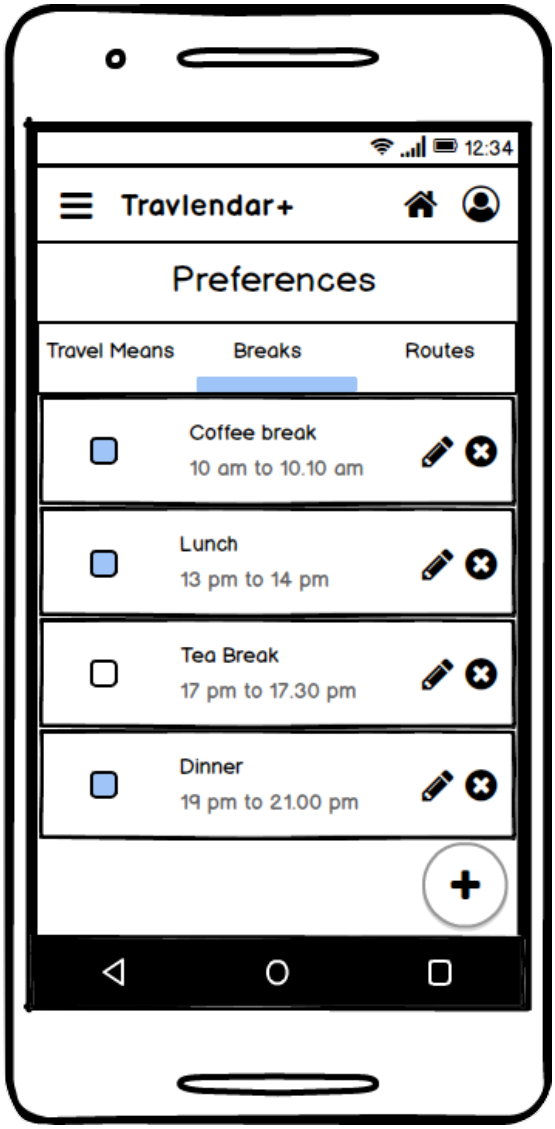

Figure 12: Break preferences page.

4.2.8 Route preferences

The route preferences page is very similar to the Travel mean preferences page. The style is the same and the opportunity to flag or unflag elements too, however there are differences. Some of the preferences in this section are mutal exclusive, so the user is prevented to select more than one of them (for instance a user can select either the shortest route or the fastest). Moreover, some selection element has customizable details, they are represented with a pencil logo on the right, and can be managed by the user to best fit his preferences (for example the user can select after which hour he does not want to use public transportations).

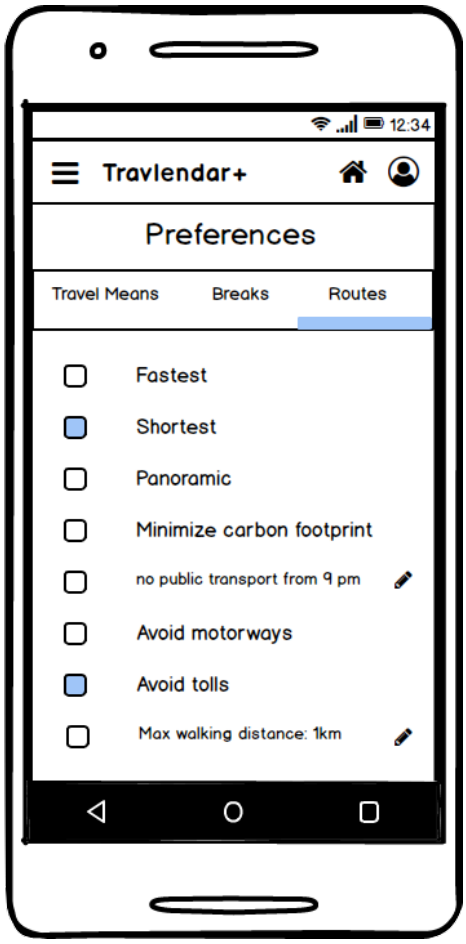

Figure 13: Route preferences page

5 Requirements Traceability

6 Implementation, Integration and Test Plan

6.1 Implementation Order

The implementation will start trying to understand if the chosen technologies are actually the best match to create the software.
To be able to determine if that is actually the case the first small goal is to create a very simple communication protocol between the Application Server and the Application Client (Android app), then integrate this with a simple database and trying to store and get information from it.
After this we will be able to enlarge the overall software by adding one small functionality at a time. At the beginning we will concentrate more on the server side and start building some EJB which incapsulates the main functions which the server will have to perform.
When all the essential features work we will start to develop the actual User Interface of the application.

6.2 Integration Testing Strategy

The natural integration testing strategy we came up with and that strictly follows the implementation order is the **Threads** approach. In particular, within a thread the strategy used to integrate and test modules will be the **Bottom-up** one. Thus, we will start by integrating and testing single portions of the modules starting from the ones which do not need any stub.
Simple and small drivers will be created in order to give inputs to the portion of each module till a complete tiny feature is completed, then the other threads will follow the same pattern until the application reaches its completion.
This global strategy will allow us have a working application very early while in the meantime anticipating the testing as much as possible, so as to minimize the cost of repair in case an error were to be found.

7 Appendix

7.1 Used software

Task	Software
Edit and compile LaTeX code	TeXmaker, TeXstudio
UML modelling	Astah Pro
UX diagram	Signavio
Mockup creation	Balsamiq Mockups 3

7.2 Effort spent

- Matteo Marziali working hours: ≈ hours
- Mirko Mantovani working hours: ≈ hours