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

Travlendar+

Design Document

V2

Mirko Mantovani (893784), Matteo Marziali (893904)

January 3, 2018

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

- **V2 on January 3, 2018:** Complete revision of component diagrams and database after implementation. Added new user interface screens on web application.

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.

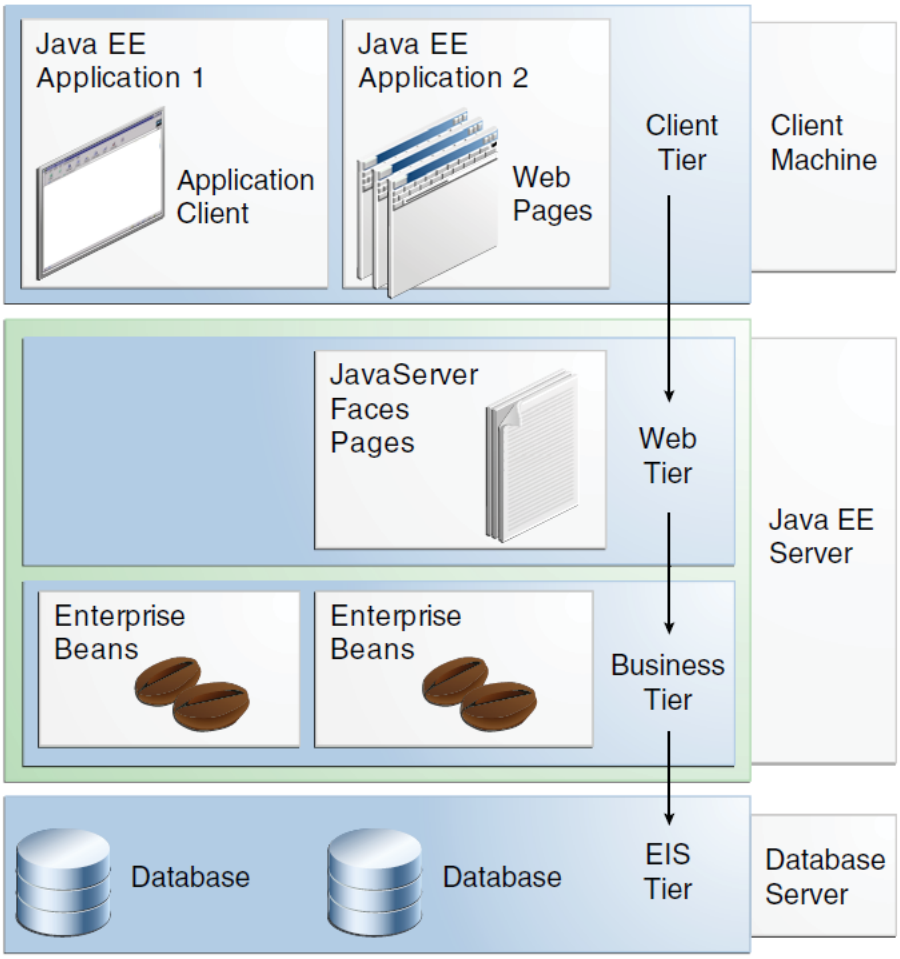

Figure 1: JEE architecture

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.

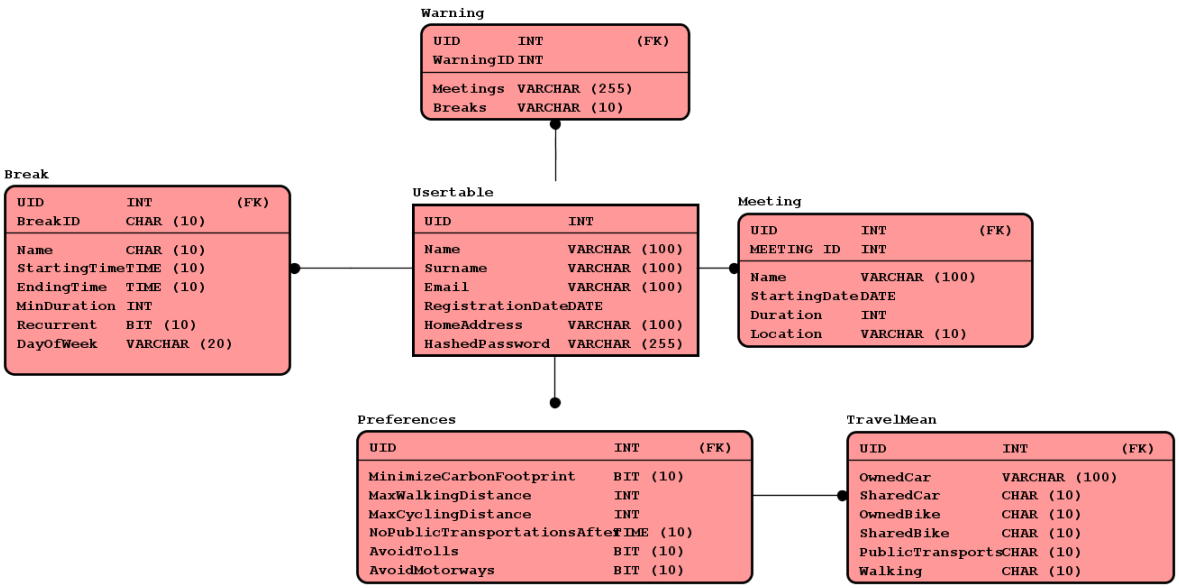

Figure 2: The E-R diagram of the database schema.

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

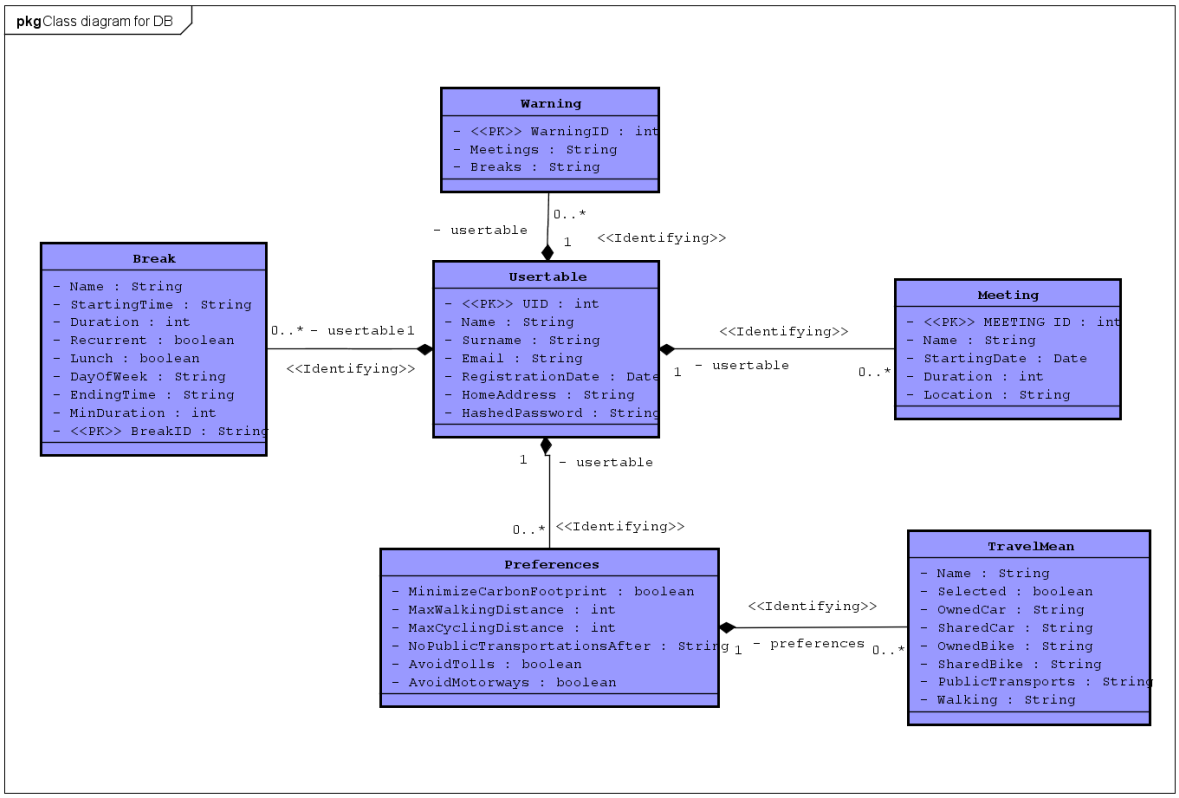

Figure 3: The E-R diagram of the database schema.

2.3.2 High-level component diagram

In order to provide an overall schema of the whole architectural structure of our application, it seems relevant to show the mappings of the main components in such a high-level representation.
At the top of the structure there is the client tier, the layer which allows the direct interation with users. This layer is composed by the user interfaces of the mobile application and the web pages of the web application.
The core of the architecture consists in the business tier, the layer in charge of providing the computations to assolve the functional requirements and to make them available through the presentation tier for the users. The main components of this tier are the Entity Java Beans of the application server and the servlets and Java Server Pages/Faces of the web server.
At the bottom of the architectural structure there is the persistence level, which essentially consists in a database. The database stores all the data available and manages to answer the queries made by the components of the business tier to retrieve information useful for computational processess.

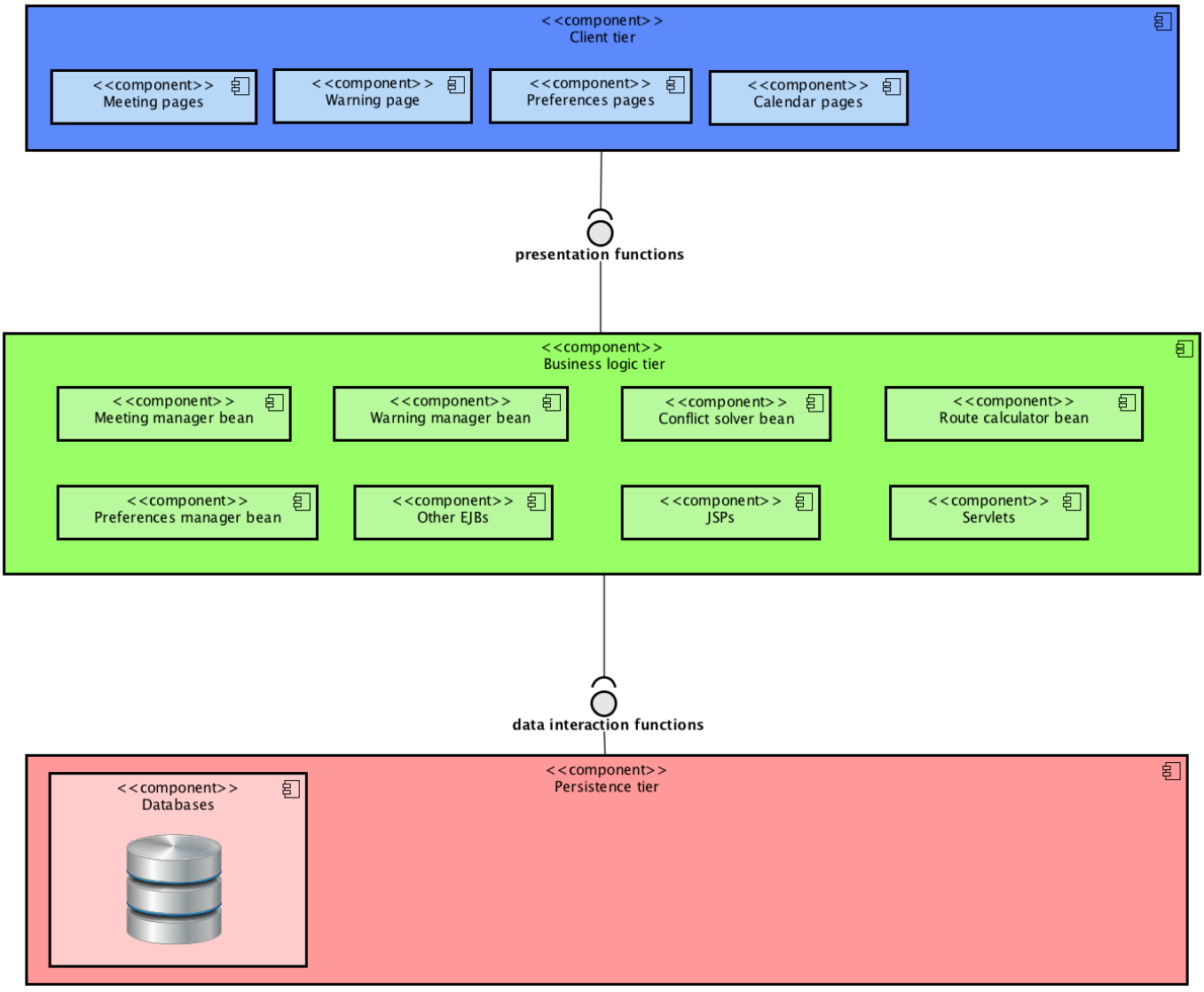

Figure 4: High-level component diagram

2.3.3 Low-level component diagram

This component diagram, as the title suggests, is designed to deeply analyze the atomic components of the application server, since they are the core of the whole architecture in terms of complexity of computations and relevance in assolving functional requirements.
Since this diagram aims to explain almost every logical component and their interactions, it is quite complex and difficult to be read and understood, for this reason we attached below some tables to further clearify both the interactions and the role of each component.

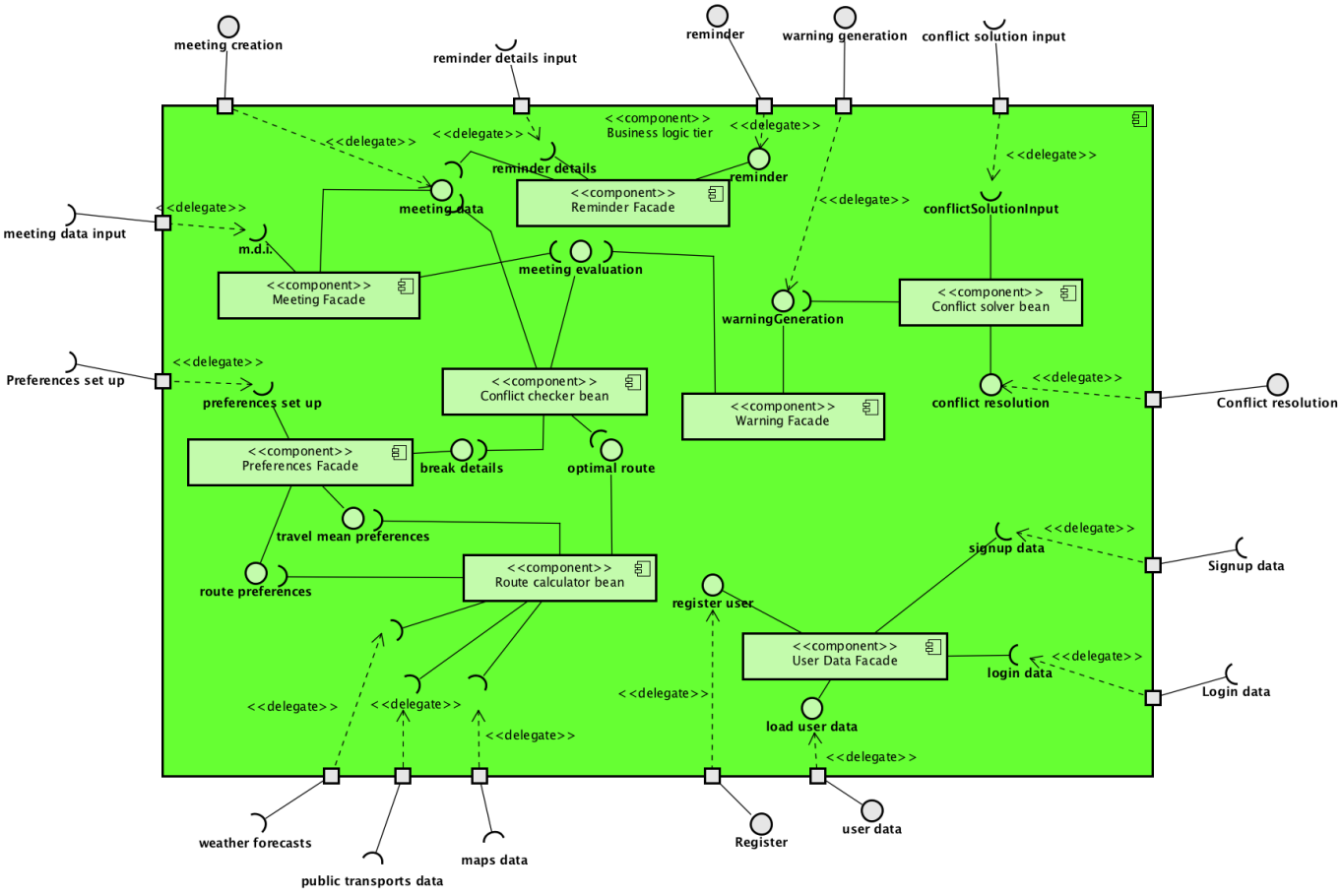

Figure 5: Low-level component diagram

Name	**Meeting Facade**
Input interfaces	• Meeting data input. • Meeting evaluation.
Output interfaces	Meeting details
Description	This is the component responsible for managing the insertion of meeting data by the user, it also provides these details to the meeting evaluator bean and to conflict evaluator bean.

Table 1: Meeting manager table

Name	**Conflict checker bean**
Input interfaces	• Meeting data • Break details • Optimal route.
Output interfaces	Meeting evaluation
Description	The Conflict checker bean has the role to evaluate if a meeting can be scheduled with respect to the adjacent meetings and the breaks. To execute this critical computation it needs information about the breaks, the new meeting and the time to reach the meetings which precede and follow it. The result of the conflict checker bean is required by the conflict evaluator to establish whether it is necessary to generate a conflict or not.

Table 2: Conflict checker table

Name	**Warning Facade**
Input interfaces	Meeting evaluation
Output interfaces	Warning generation
Description	This component represents the bean which is responsible to generate a warning and submit it to the user if the meeting evaluator notify a conflict.

Table 3: Warning manager table

Name	**Conflict solver bean**
Input interfaces	Warning generation
Output interfaces	Conflict resolution
Description	The conflict solver is the component that manages the resolution of conflicts, thus it is called by a new warning and his functions are to ask the user whether ignore or modify the meetings involved in a conflict and to apply his choice.

Table 4: Conflict solver table

Name	**Preferences Facade**
Input interfaces	Preferences set up
Output interfaces	• Route preferences • Travel mean preferences • Break details
Description	This is the component designed to assolve all the functions related to the preferences. Hence, it manages the insertion, update and deletion of the preferences about routes, travel means and breaks, furthermore it provides preferences details to other components when it is necessary.

Table 5: Preferences manager table

Name	**Route calculator bean**
Input interfaces	• Travel mean preferences • Route preferences • Weather forecasts • Public transport data • Maps data
Output interfaces	Optimal route
Description	This is the component designed to assolve all the functions related to the preferences. Hence, it manages the insertion, update and deletion of the preferences about routes, travel means and breaks, furthermore it provides preferences details to other components when it is necessary.

Table 6: Route calculator table

Name	**User Data Facade**
Input interfaces	Login data, signup data
Output interfaces	User data, registration
Description	This is the component which manages the user signup process, from the insertion of personal data, through their storage on the database, to the account verification.This component is also dedicated to all the login and signup steps. When the user insert in the apposite area his name and password, this component takes the data and provides the computations needed to verify the credentials and load the user's content.

Table 7: Login manager table

Name	**Reminder manager bean**
Input interfaces	• Meeting data. • Reminder details.
Output interfaces	Reminder.
Description	This is the component in charge of allowing the creation of a reminder by the user. Because of the reminder addition button is in every meeting page, it will be the Meeting manager the component which eventually will call the reminder manager. Then, the reminder manager will ask the user for the reminder details such as a schedule time and date before managing to generating the proper reminder in the system.

Table 8: Reminder manager table

2.4 Deployment View

The deployment diagram represent our intention to express a complete view of the system in terms of adopted hardware components, communication languages and software components distribution among them.
Among the huge number of services, we have decided to use GlassFish as web and application server and to adopt the DerbyDB database management system.
As far as the software components are concerned, the application server has the Entity Java Beans and the Java Persistence API to allow the interaction with the database, oppositely the web server has the servlets and the Java Server Pages.
Since we have initially thought about a web application, the User mobile phone could be omitted, however we have decided to insert it in our architecture to make the system ready to future improvements, such as, eventually, an app. dedicated to mobile devices.
The diagram is filled with a grey box named 'External services servers' to design also the interaction between our system and the systems of other institutions such as Google Maps or the municipality of Milan etc. which can provide basical data to reach our goals and to implement Travlendar+ functions.

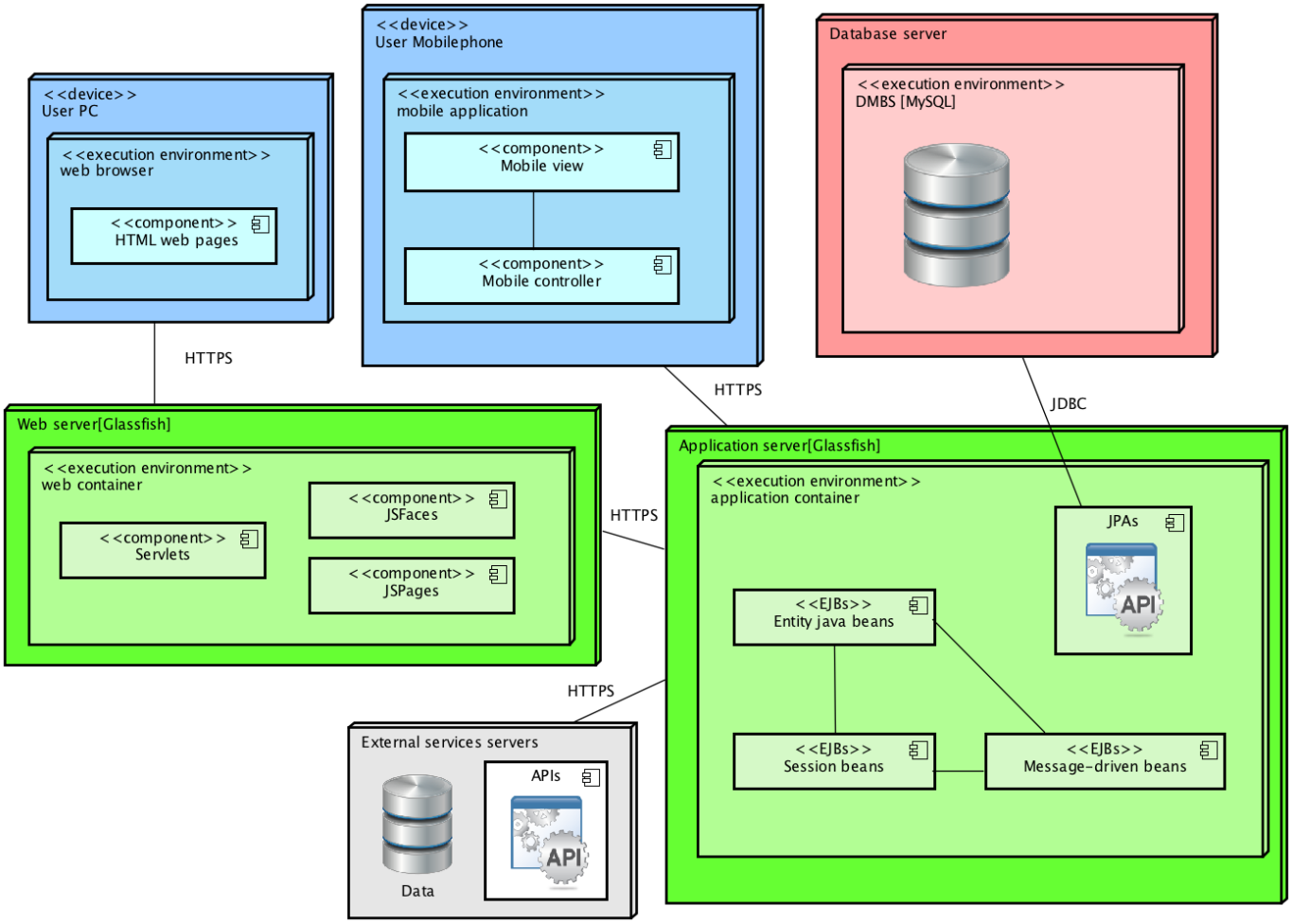

Figure 6: Deployment diagram

2.5 Runtime View

2.5.1 Signup/login sequence diagram

The Signup and Login runtime view includes, as the name suggests, both the signup and the login (and also the password recovery), since they are intended related concepts with a slight impact on the application behaviour.
A registered user who wants to log in himself must fill the login form until he insert correctly his username and password, at this point the Login manager bean manages to identify him and then to load his data.
On the other hand, when an unregistered user shows the intention to sign up himself, the Signup manager bean is called and submits the user a registration form, and, once the user has completed and sent back it, manages to verify the consistency of the inserted data.
If a registered user has forgotten his password, the Login manager component is called and requires the user email to send him a link where a new email can be created, then the login can eventually be performed by the user.

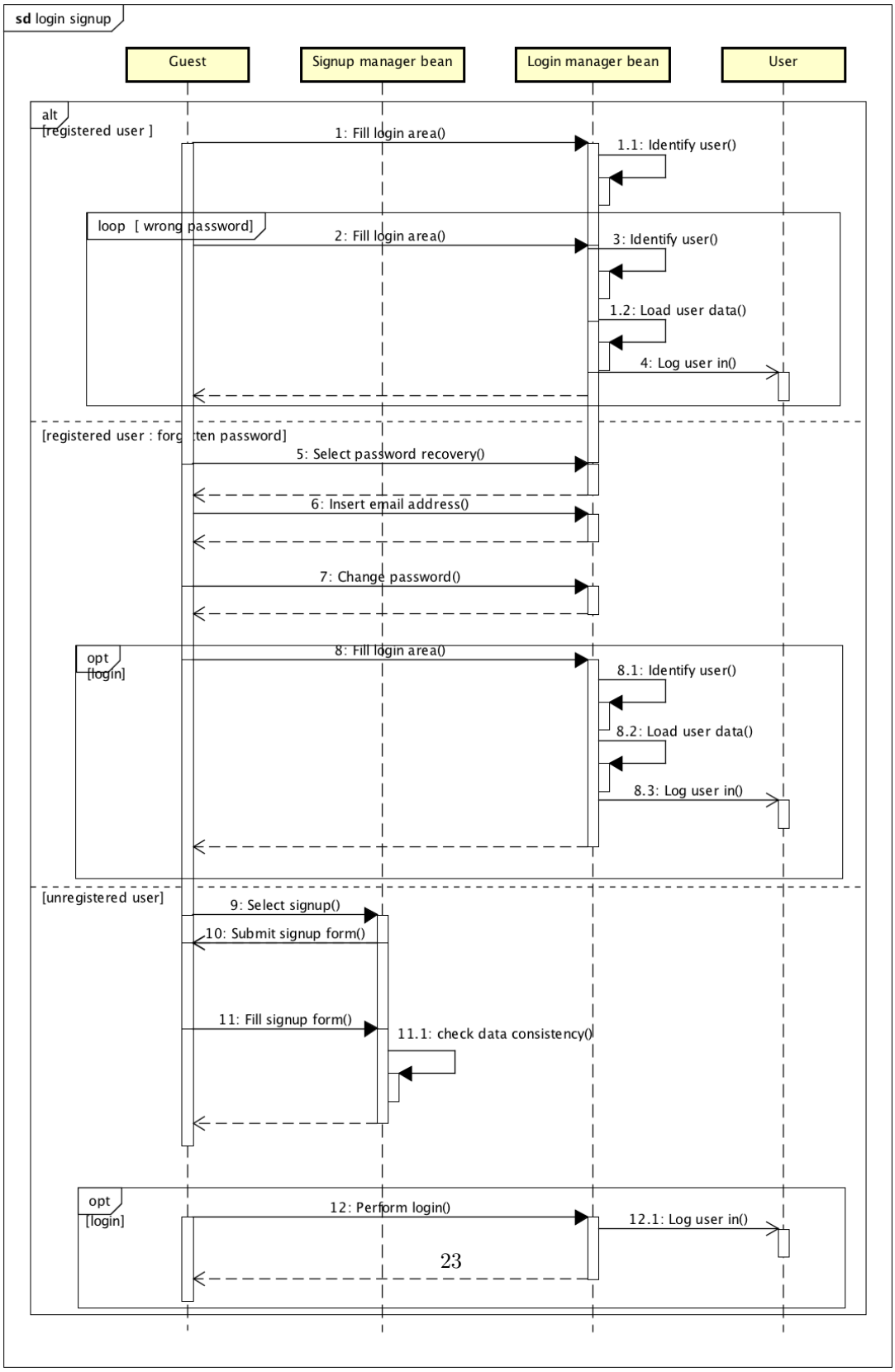

Figure 7: Signup/login sequence diagram

2.5.2 Meeting creation sequence diagram

The meeting creation runtime view, explained through an apposite sequence diagram, summarize all the computations needed to allow the creation of a meeting.
This scenario starts when a user select to create a meeting, this intention is dispatched by the application server to the Meeting manager bean, which is in charge of submitting a meeting creation form to the user and holding its resubmission.
Taken the meeting details, the Meeting manager sends them to the conflict checker, this component must control whether a meeting is feasible or not, considering both the breaks set by the user and other meetings.
To assolve this function, this bean needs thee break details from the preferences manager bean and some routes from the google maps server in order to evaluate the time among adjacent meetings.
Collected the information, the conflict checker computes to find overlaps and if it is not the case, the negative verification outcome reaches the user, who is notified about the correct insertion. In case overlaps are present, if they are caused by conflicts with breaks that can be flexibly rescheduled during their range not to be overlapping the meeting will be accepted without any consequence. Otherwise, if either this is not the case or the conflict is beetween meetings, the conflict checker will generate a conflict and notify the user through the meeting manager.

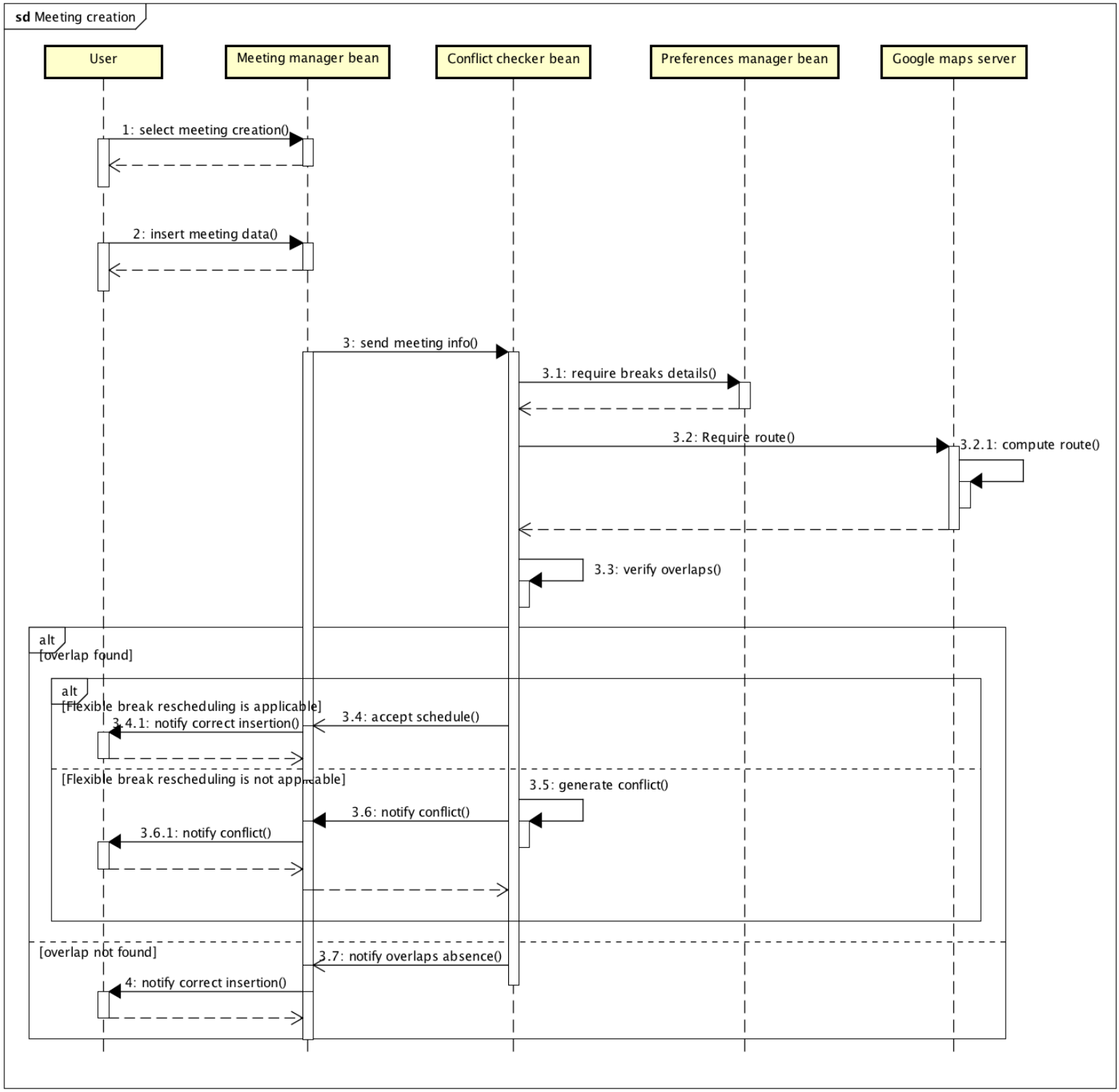

Figure 8: Meeting creation sequence diagram

2.5.3 Conflict management sequence diagram

The conflict management runtime view refers to every situation in which a conflict is generated even if in the provided sequence diagrams the computations start with a meeting insertion. We have already discussed the positive outcome that simply ends with a notification to the user, now we want to deeply analyze the presence and the management of a conflict.
When a conflict is found, the conflict checker notifies the warning manager which generates a warning in the system and notifies the conflict solver component, that is in charge of asking the user in which way he/she intend to solve the conflict and is in charge of applying the user choice.
Firstly, the conflict solver manages to ask the user whether ignore the conflict or modify the involved meetings to solve it. Then, it holds the user decision and computes to make it effective in the system.
If the conflict is ignored, it is deleted and the system behaves like it isn't any conflict. Otherwise, if the conflict is solved by the user, the conflict solver sends an updated calendar to the meeting evaluator which computes to verify overlaps.
If conflicts are found this cycle of operations is repeated, else the user is notified that his schedule is consistent.

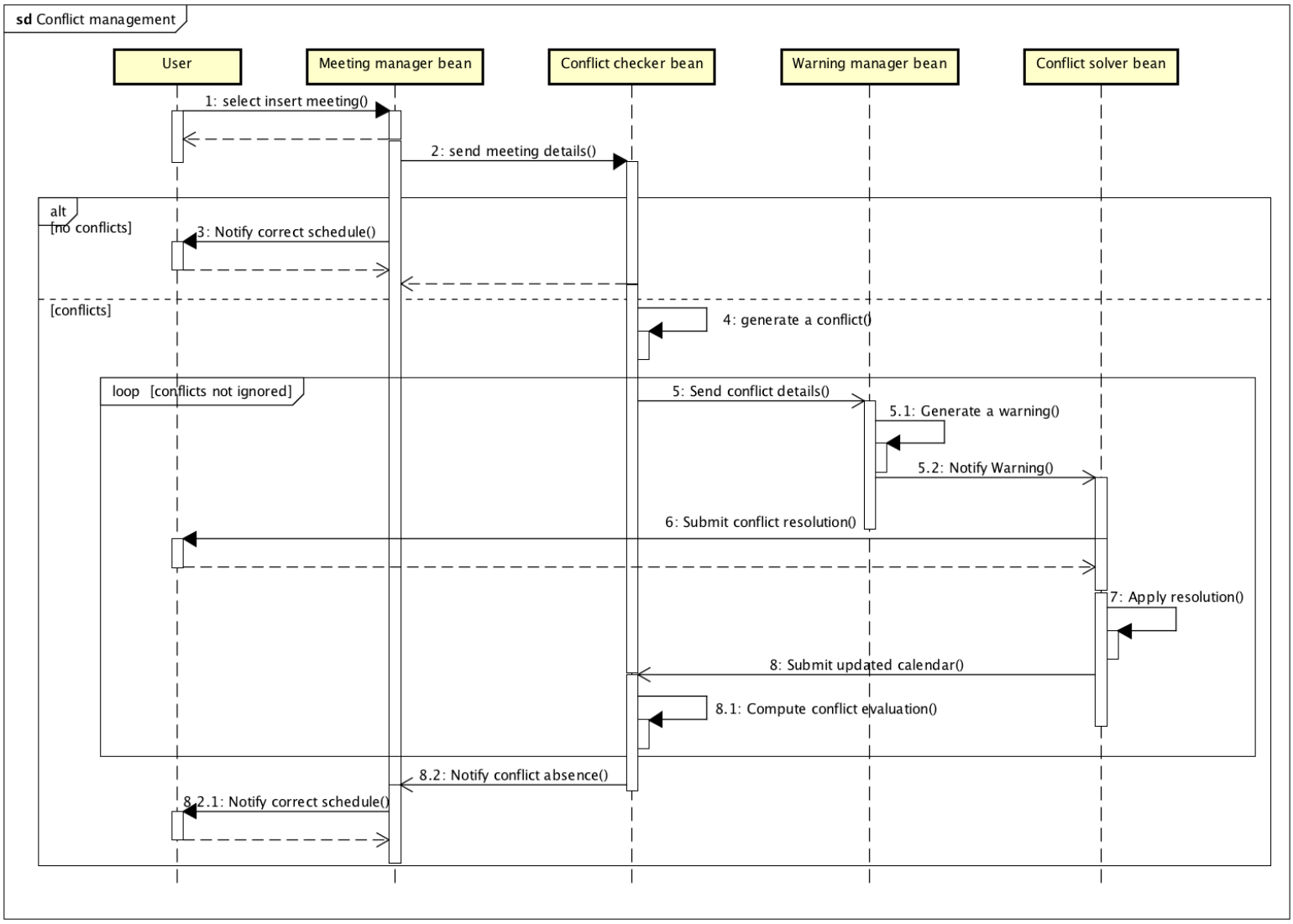

Figure 9: Conflict management sequence diagram

2.5.4 Break creation sequence diagram

The break insertion runtime view is conceptually similar to the meeting insertion even if it involves less components and requires less controls.
When the user express the intention to insert a break, the preferences manager bean is called and manages to submit the user a form to get the break details. At this point, the preferences manager sends this details to the meeting evaluator which plays the role to search conflicts by verifying overlaps.
If it finds conflicts, it generates a conflict in the system otherwise nothing is done and in both cases the user is notified.
Please note that in the break insertion the only possible conflict refers to a situation where the new break overlaps completely one or more meetings which has the only implication of generating a conflict in the system.

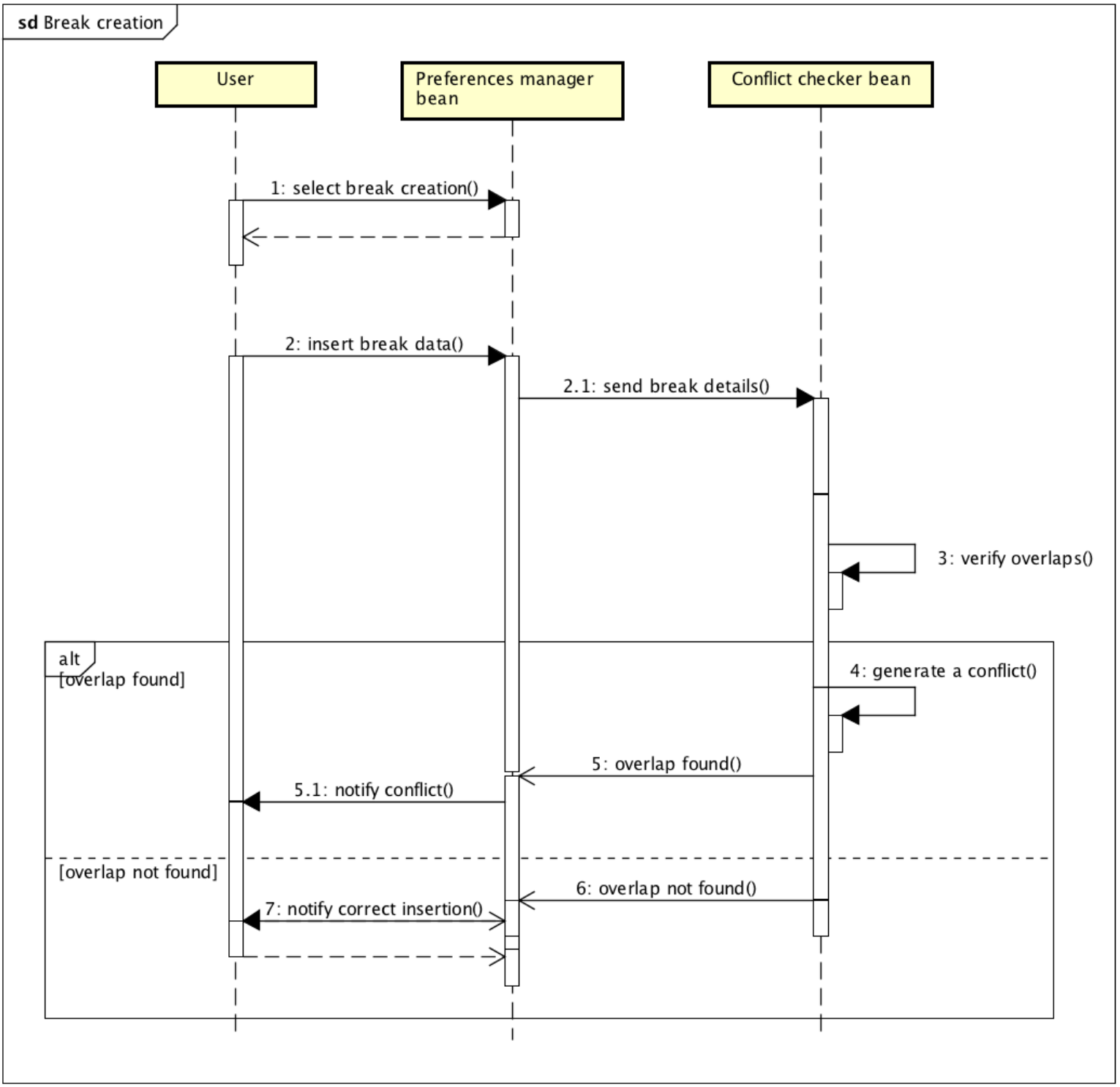

Figure 10: Break creation sequence diagram

2.5.5 Preferences set up sequence diagram

The preferences insertion runtime view aims to provide a representation of how a user can set up all the customizable settings of Travlendar+.
Firstly, it is relevant to notice that the breaks preferences are considered in a different sequence diagram.
This said, the user still can activate or deactivate travel means, to allow or prevent them to be used in the routes for his meetings, clearly at least one travel mean must be activated, this explains the travel means deactivation loop and its condition.
Moreover a user can eventually activate or deactivate routes, in the sense that a user can decide to have the fastest route, the shortest route etc. As far as the routes are concerned, there arenn't constraints referring to have at least one route activated, indeed in that case it is expected that a default route is adopted.

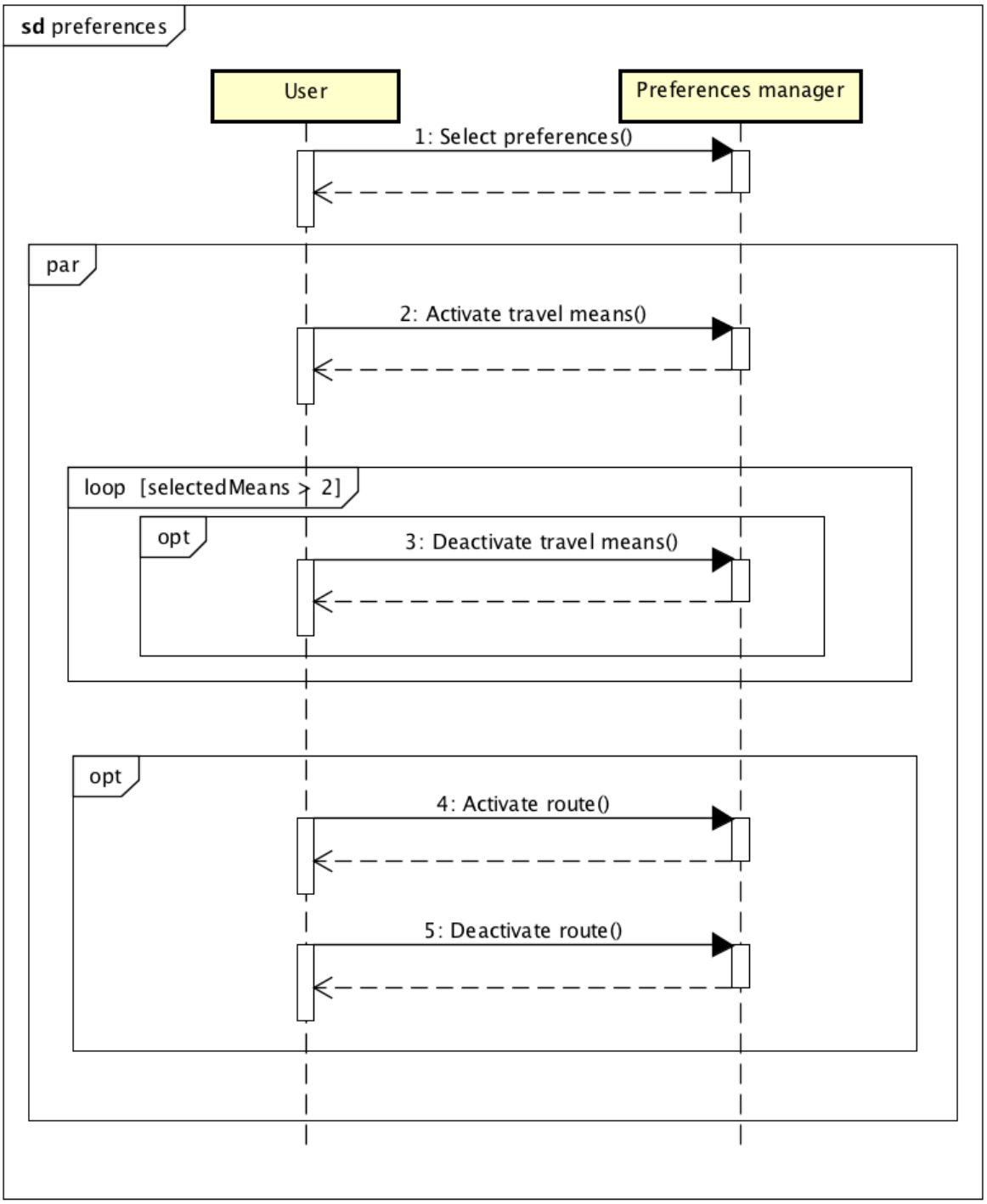

Figure 11: Preferences set up sequence diagram

2.6 Component Interfaces

This sections includes further details on the interfaces between different macro-components and components of the system. Also, Subsection **??** is devoted to illustrate some relevant details about the interfaces needed to use and interact with each component of the Application Server, accessible both by other Application Server components and by other components of the system.

2.6.1 Database - Application Server

The Application Server is the only one that can access the Database directly; this is done through the Java Persistence API mapping between objects and actual relations.

2.6.2 Web Server - Web Browsers

The interactions between the client's browsers and the Web Servers are based on the HTTPS protocol, with standards GET and POST methods.

2.6.3 Application Server - Web Server and Clients

The communication between Application Server and clients happen with the HTTP.

2.6.4 Application Server - External Systems

The Application Server must interact with external APIs like the Google Maps one, which will provide information about travels.

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
```

```
39          IF nextMeeting.startTime-newMeeting.startTime+
40          prevMeeting.duration < nextTime
41          THEN
42              createWarning(newMeeting, nextMeeting)
43
44          END-IF
45      END-IF
46  END-IF
47
48  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:

```
1   BEGIN checkBreakConflict(Meeting newMeeting)
2
3   //the array breaks will contain all the breaks which
4   //overlaps with the new meeting which is being added
5   Break breaks[ ]:= READ from DB
6
7   //A break has a startingTime, and endingTime, and a
8   //minDuration. This allows a break to actually happen
9   //within the limits given by startingTime and endingTime
10  //and to have a minumum duration of minDuration, the
11  //breaks can shift in the allowed range of time
12
13  FOR i=0 TO breaks.length-1 DO
14
15    //overlappingMeetings array will contain all and
16    //only the meetings whose schedule is overlapping
17    //with the i-th break, this means that either the
18    //meeting startTime is betweeen the break
19    //startingTime and endingTime or the meeting
20    //startTime + its duration is.
21    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.

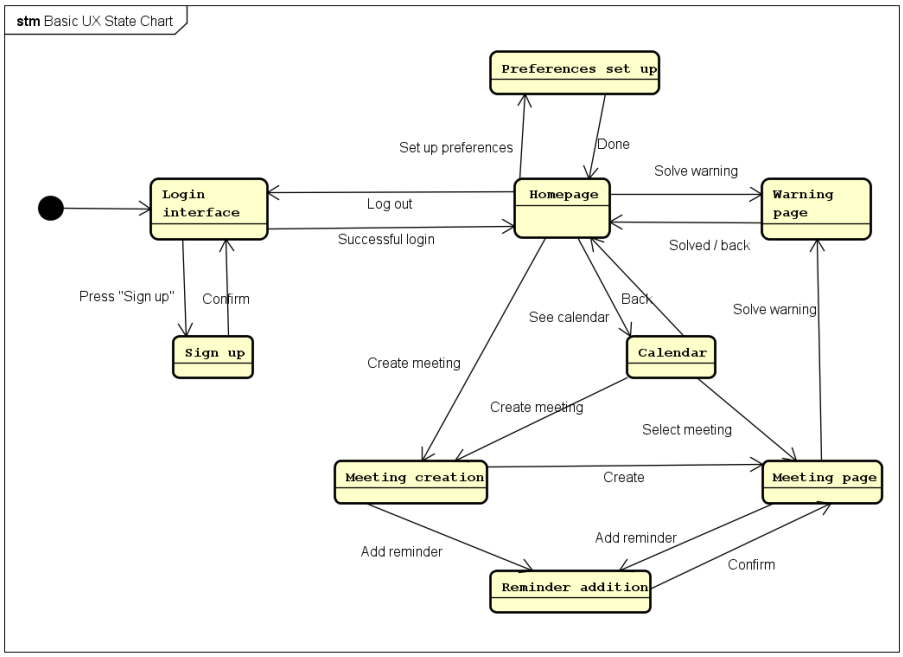

Figure 12: The initial UX diagram

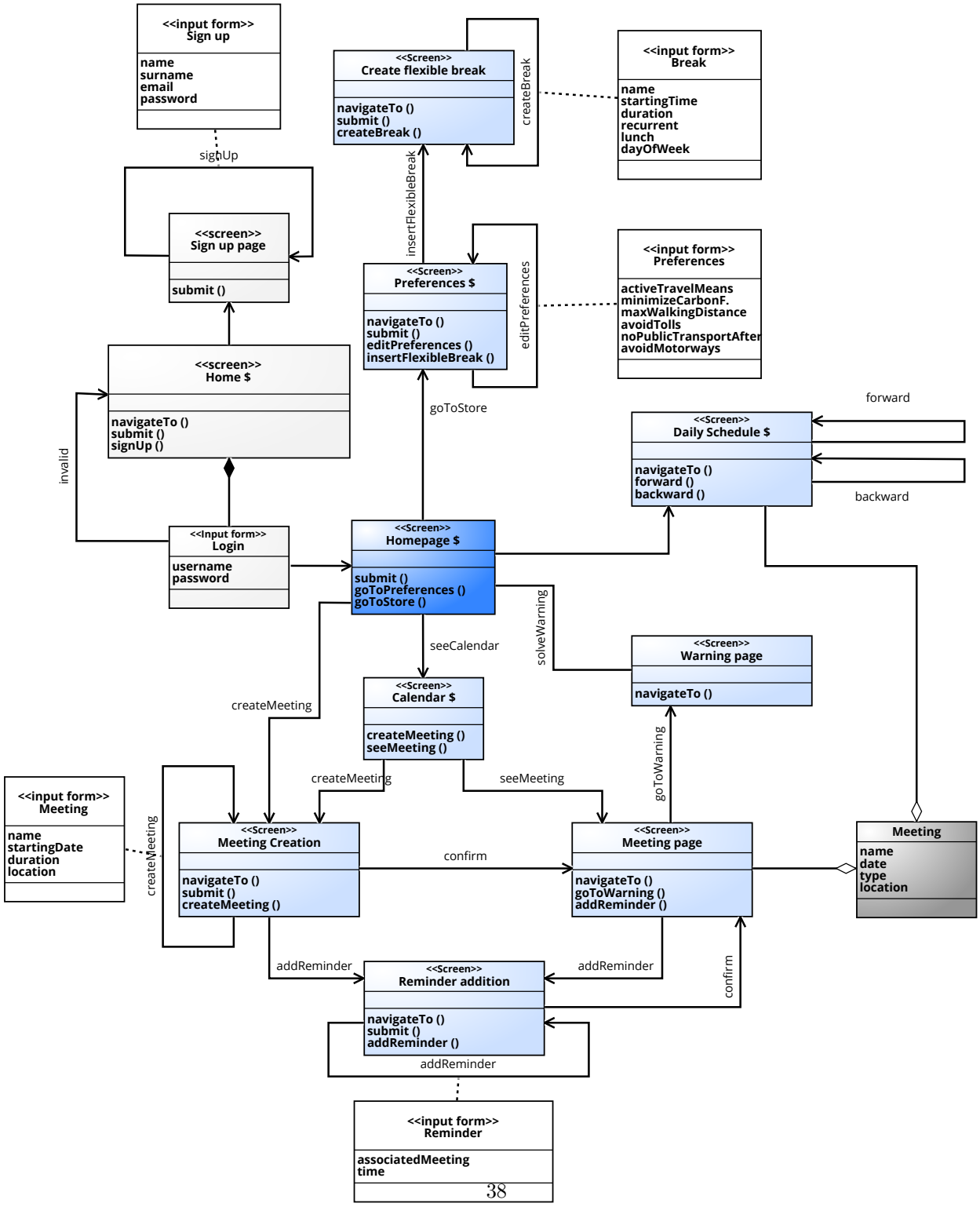

Figure 13: The complete User Experience diagram of the application

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 14: Travlendar+ homepage on the mobile application

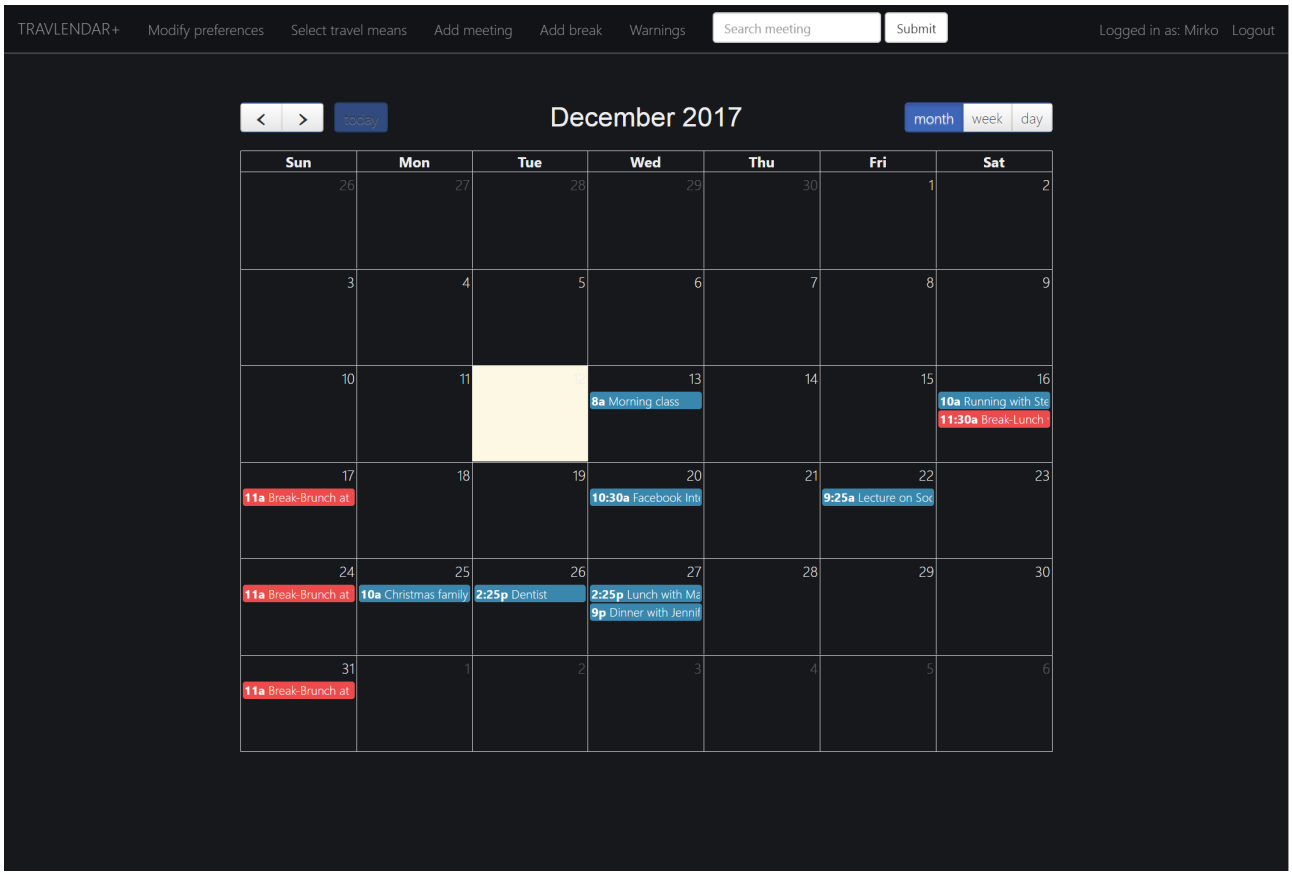

Figure 15: Homepage screen on the web application

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.

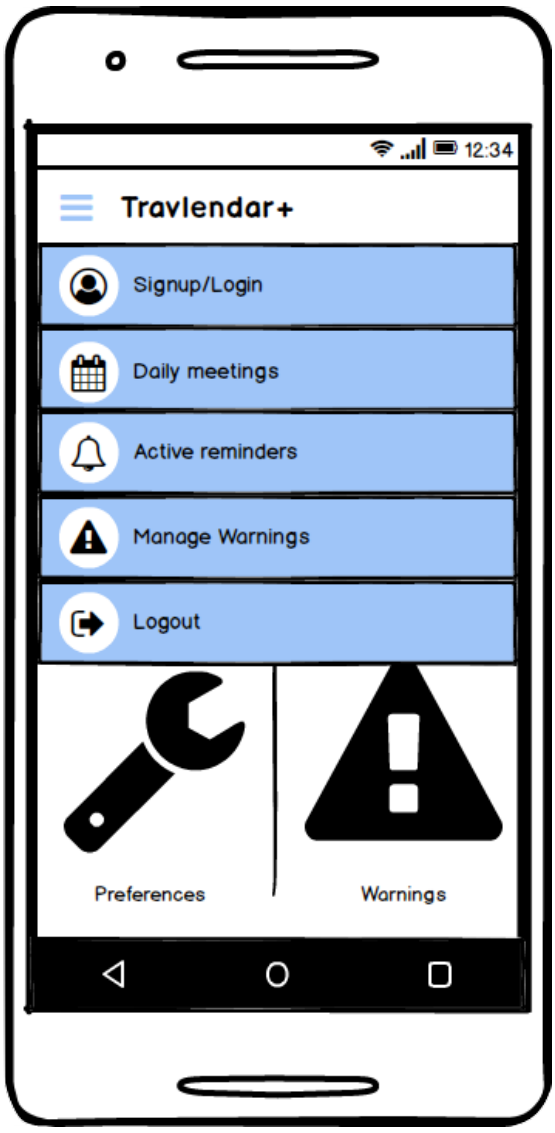

Figure 16: Quick access menu on the mobile application

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.

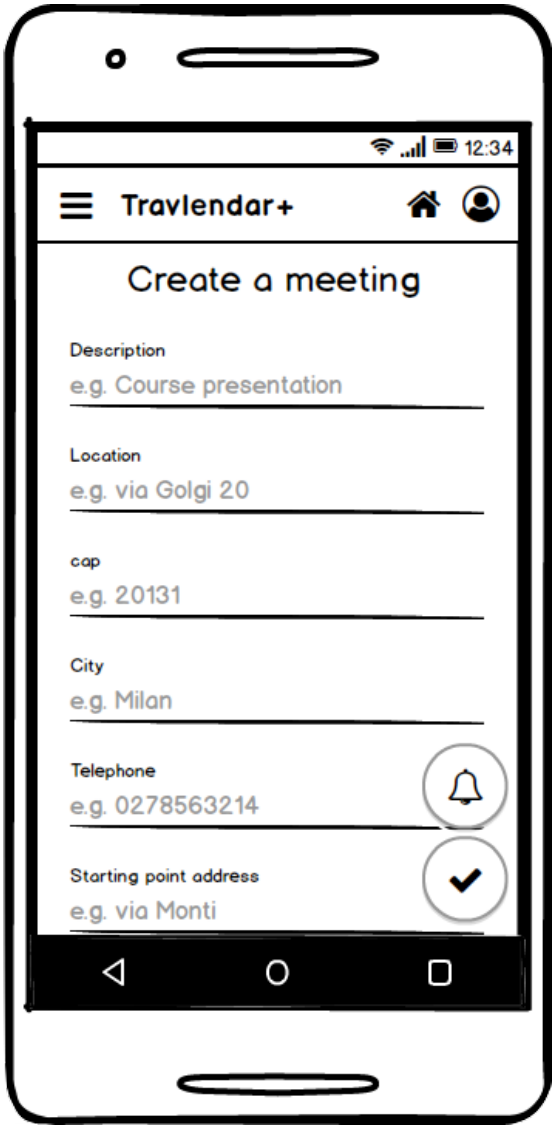

Figure 17: Meeting creation page on the mobile application

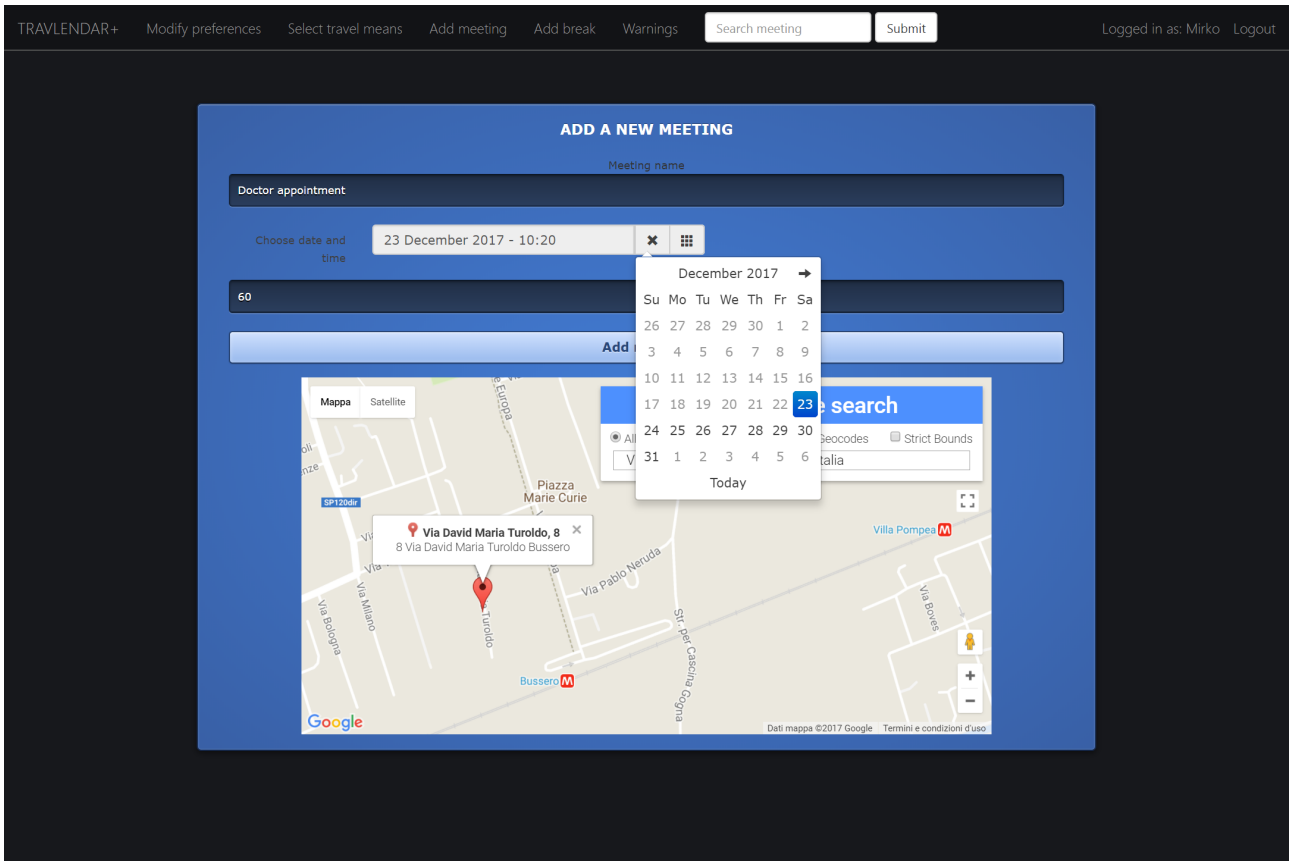

Figure 18: Meeting creation screen on the web application

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.

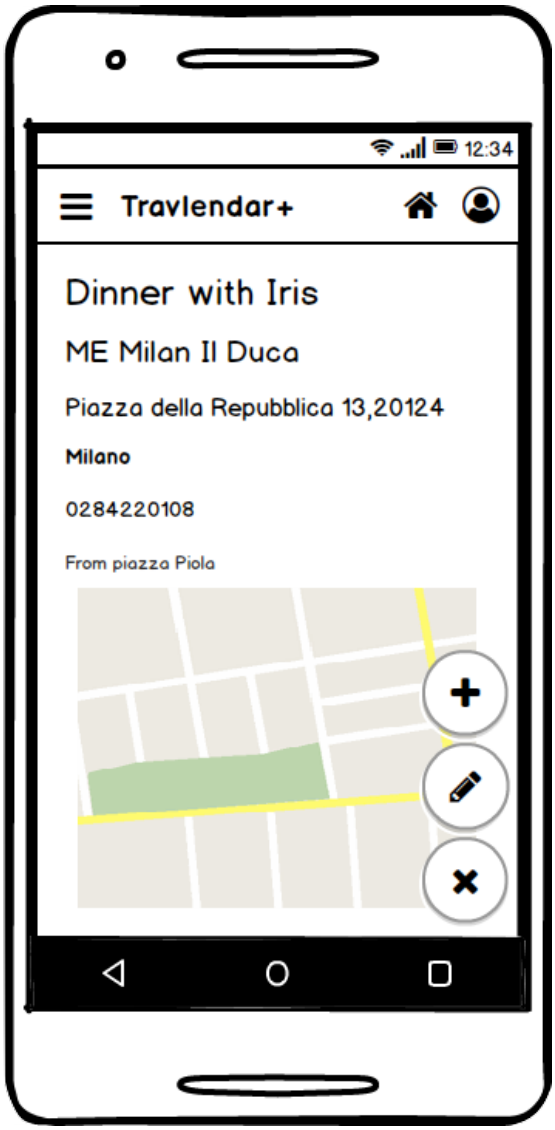

Figure 19: Meeting view page on the mobile application

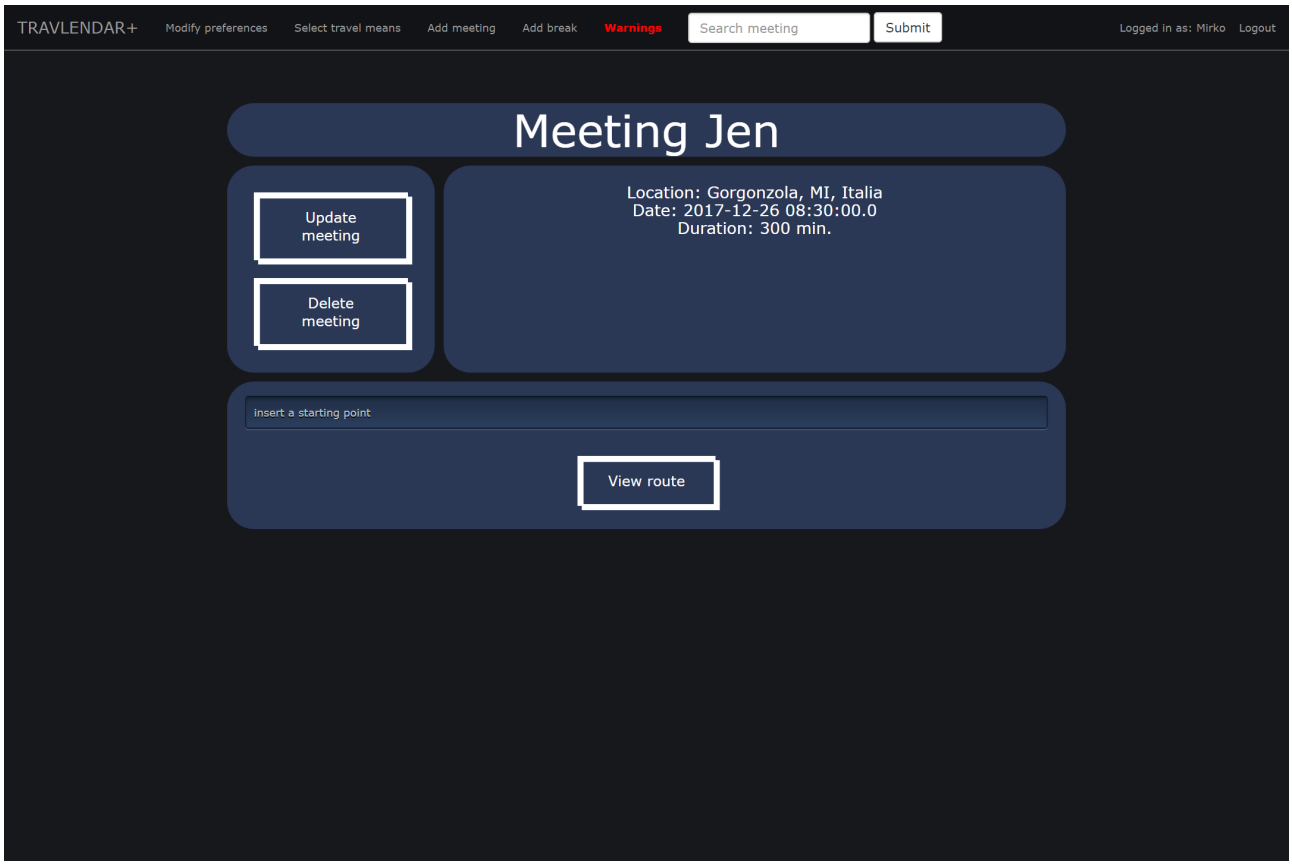

Figure 20: Meeting screen on the web application

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.

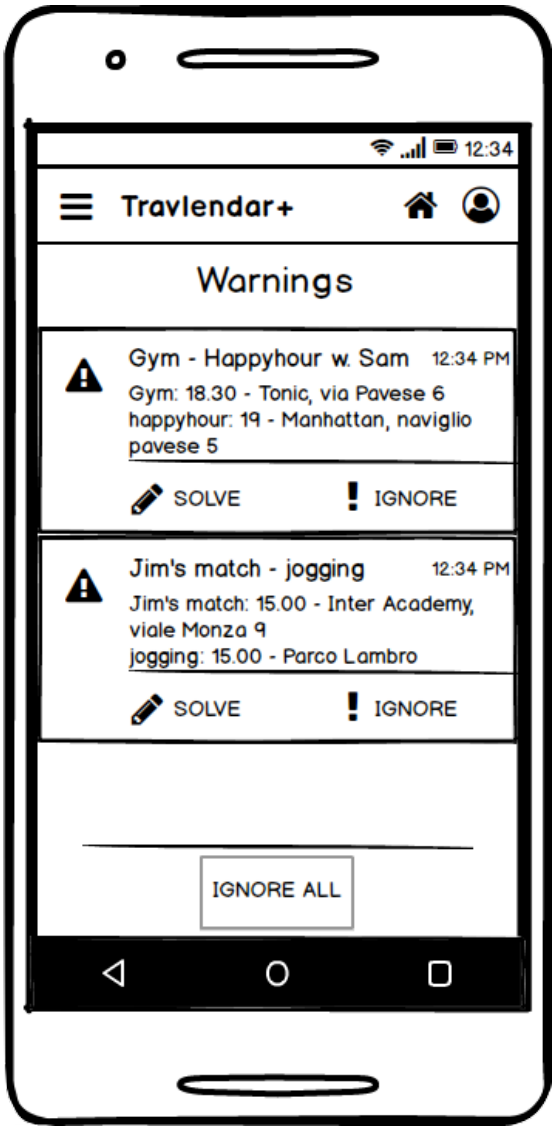

Figure 21: Warnings page on the mobile application

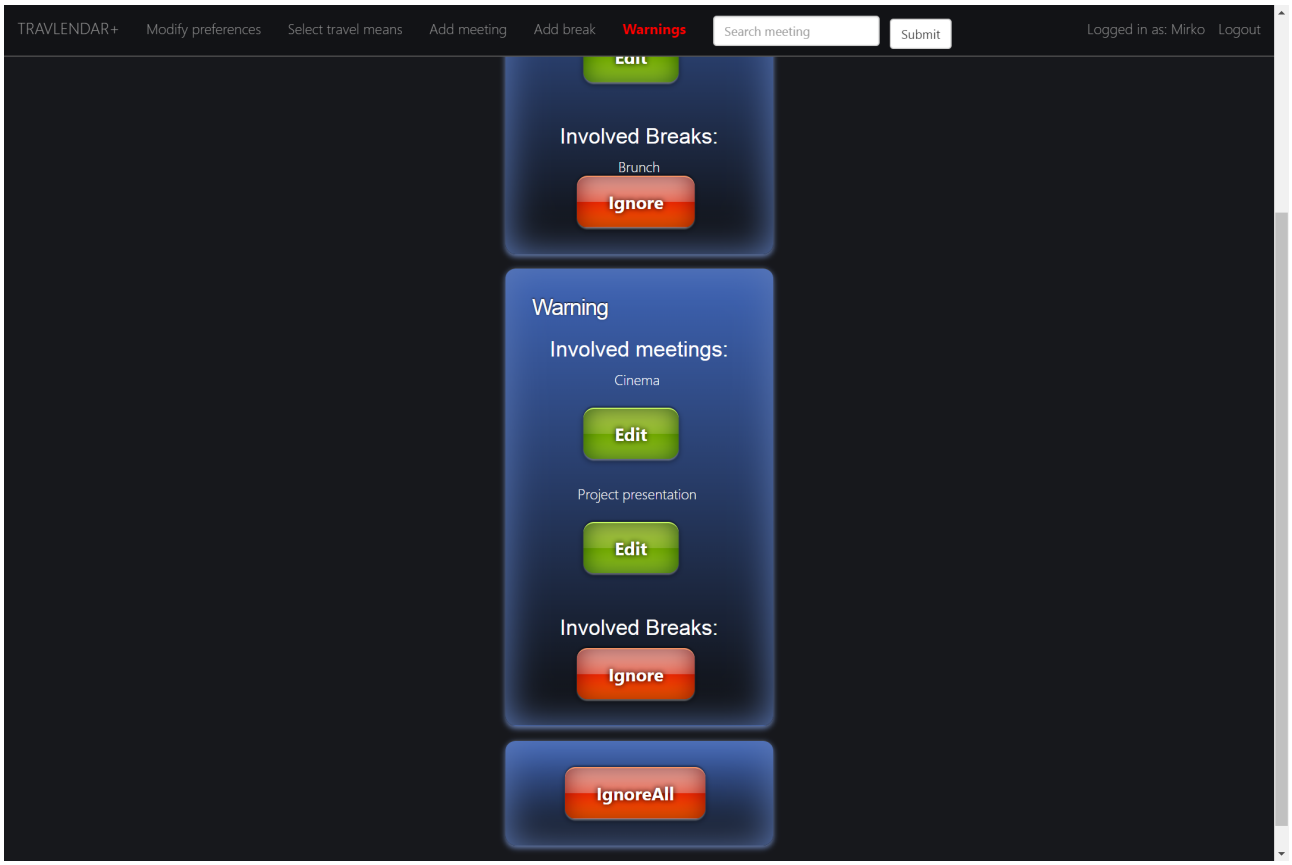

Figure 22: Warnings screen on the web application

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.

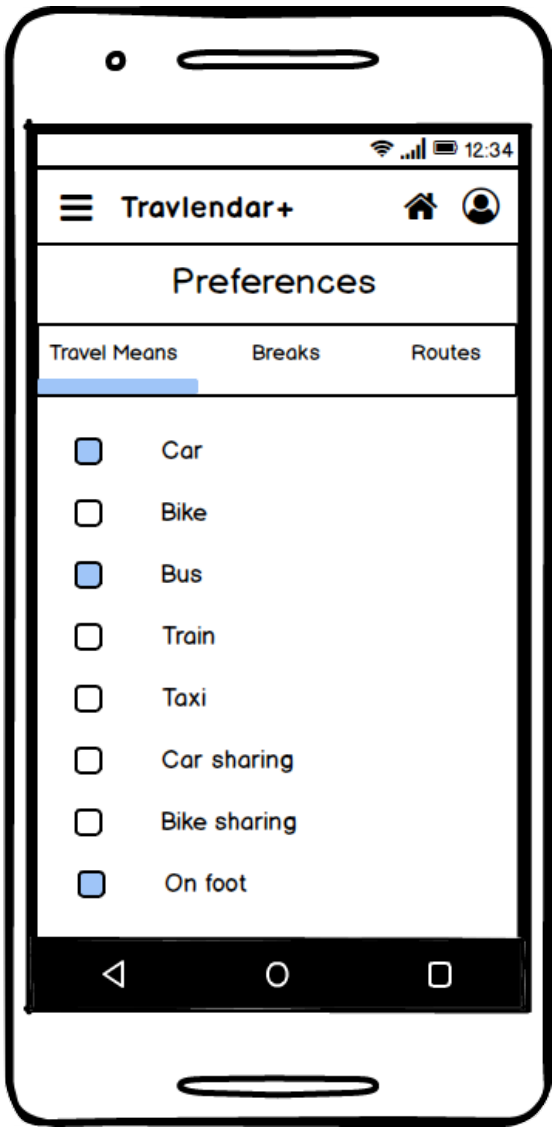

Figure 23: Travel mean preferences page on the mobile application

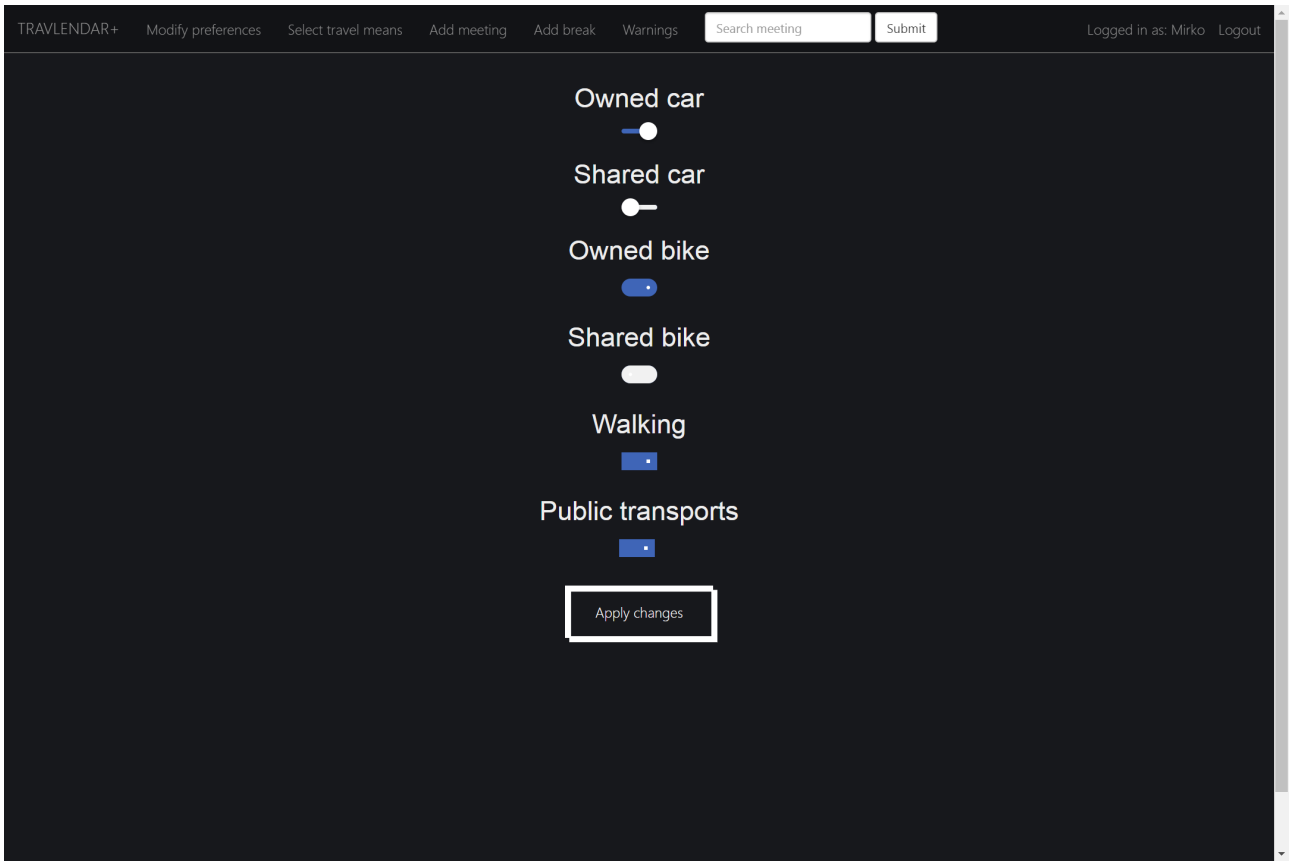

Figure 24: Modify Travel means screen on the web application

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.

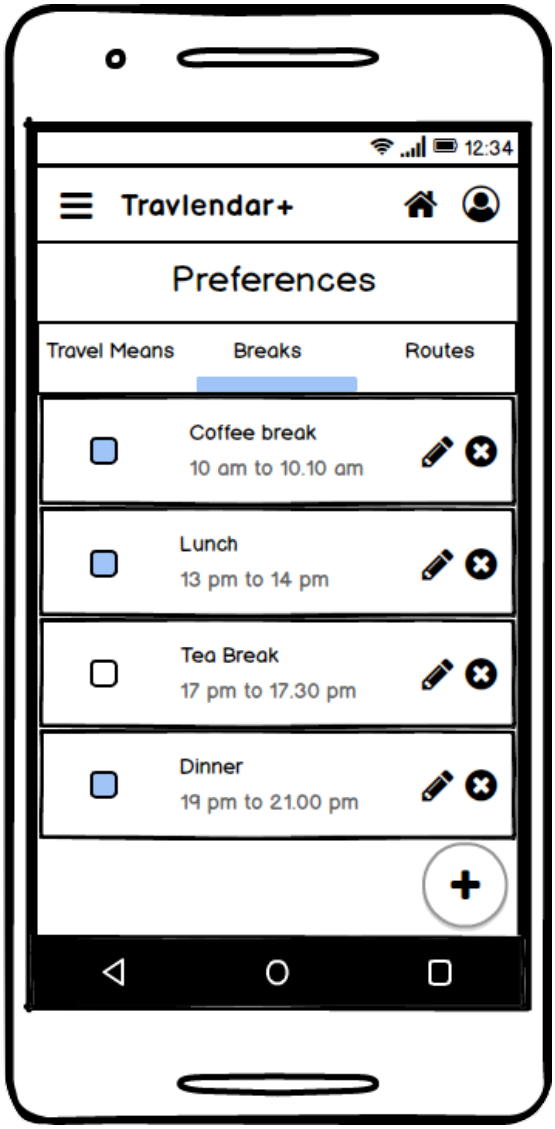

Figure 25: Break preferences page on the mobile application

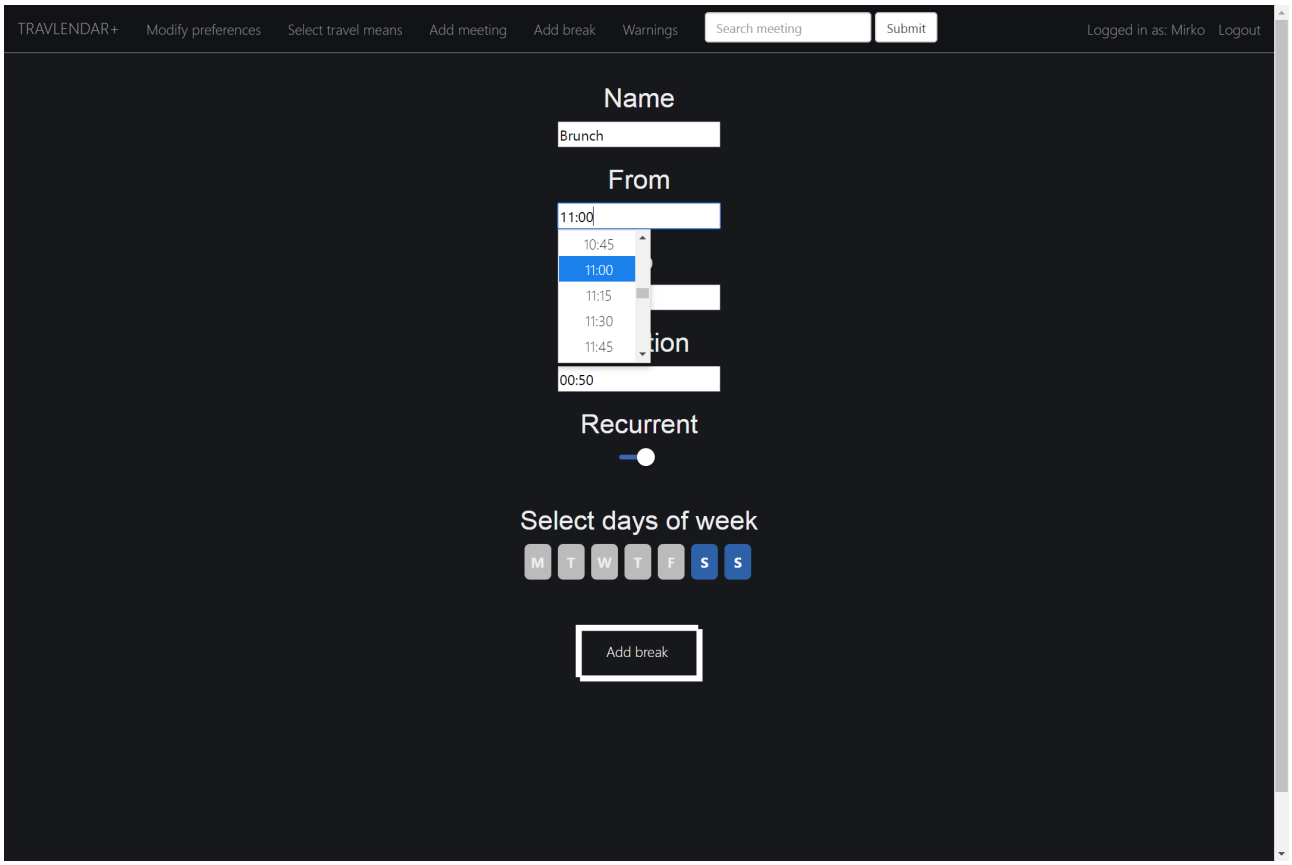

Figure 26: Add break screen on the web application

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).

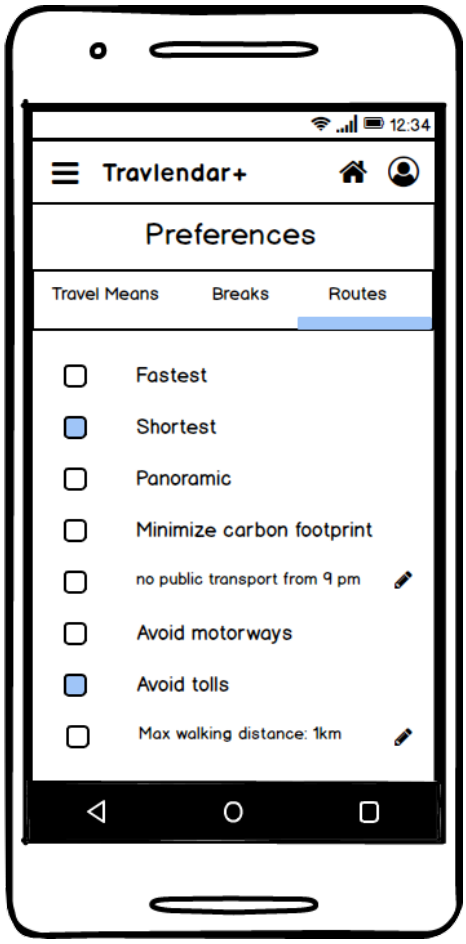

Figure 27: Route preferences page on the mobile application

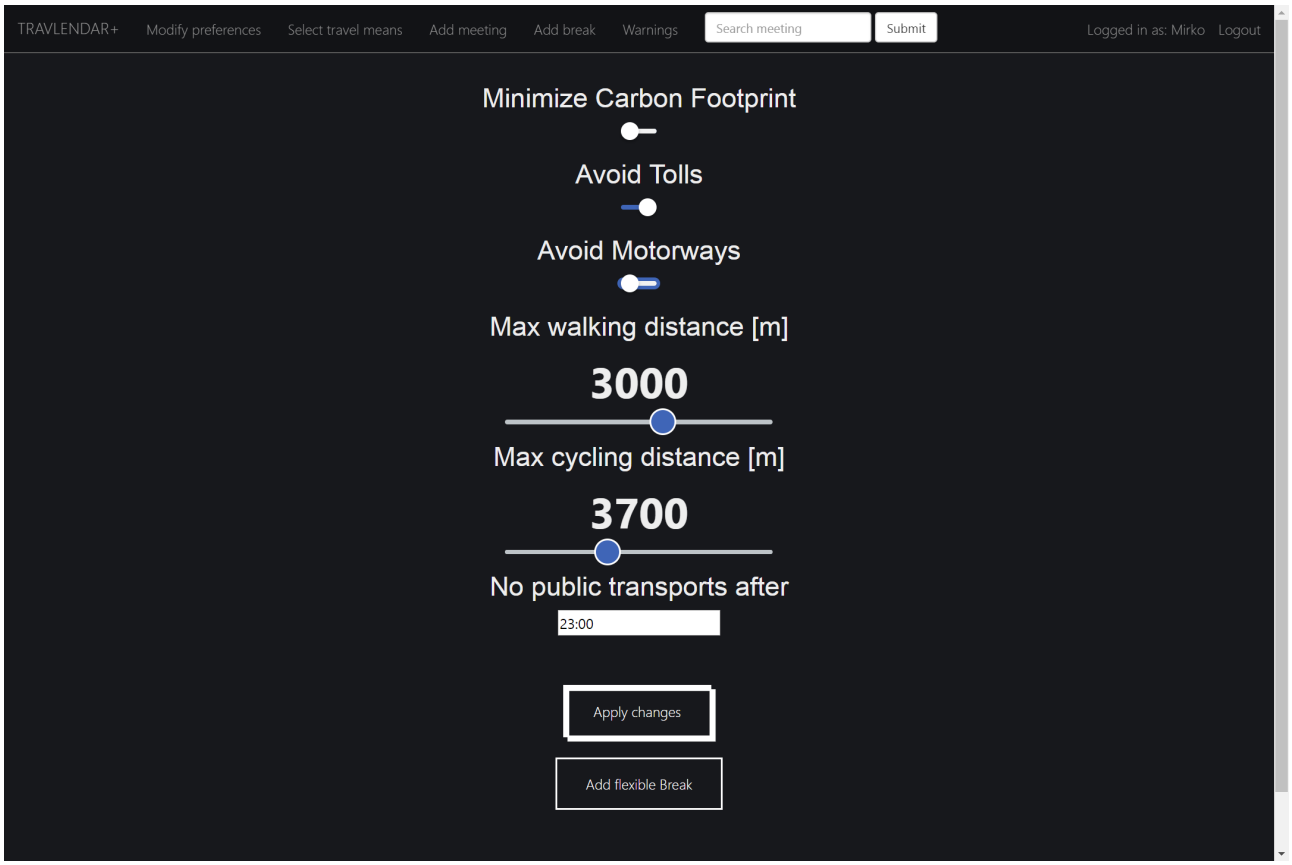

Figure 28: Modify Route preferences screen on the web application

5 Requirements Traceability

In order to have a simplified and schematic view over the designed components that are going to satisfy the requirements and provide the functionalities which were identified in the RASD document we define a formal mapping that explicitely correlates the Goals and functions with one or more Components that should be able to achieve the goal and create the functionality.

Goal	Functions	Components
[G1] Memorizing and organizing both events and appointments on the calendar.	**[F2] Meeting creation**: This is the most important function of the app, it allows to generate an event related to an appointment. It requires the user to define all the details such as date, time, location, starting point, preferences etc.	Table 1
[G2] Being sure to reach the appointment location in time avoiding delays.	• **[F6] Reminder management**: The applications allows users to set up reminders for a certain Meeting in order for the user not to be late for it. • **[F8] Update meetings**: This function is both basic and relevant, it allows the user to customize his meetings after their creation. In other words, through this functionality the user can modify each one meeting details, even in case of a warning is generated.	• Table 6 • Table 1
[G3] Being sure not to schedule overlapping meetings.	• **[F9] Warnings management**: In case a warning is generated by the system due to a possible overlap among two or more meetings, the user must solve the warning. In other words, the user has to decide wheter he wants to ignore the overlap notification or he intend to modify some meetings to be sure that he can reach and participate to all his appointments. • **[F4] Delays management**: If the app had noticed, according to the estimated travel time, that the user is in late, and he previously had inserted the email address of the meetings participants, Travlendar+ would notify them about the delay.	• Table 2 • Table 4 • Table 3

Table 9: Traceability table

[G4] Being able to use only travel means that fit with user preferences.	**[F3] Preferences set up**: An important feature of Travlendar+ consists in allowing the user to filter out specific routes depending on some constraints about the travel, or to set break-dedicated time slots.	Table 5
[G5] Preventing the user to forget an appointment.	**[F7] Recurrent events management**: The smartest function Travlendar+ will offer; it consists in allowing the user to select events to be rescheduled periodically just creating one meeting. Done this choice, the app. Automatically manages to reschedule the specific meeting according to the period that the user establish, for instance one week, one month.	Table 8
[G6] Allowing the users to modify appointment schedules.	**[F8] Update meetings**: This function is both basic and relevant, it allows the user to customize his meetings after their creation. In other words, through this functionality the user can modify each one meeting details, even in case of a warning is generated.	Table 1
[G7] Notifying other people involved in a meeting about user eventual delay.	**[F5] Route generation**: the main hidden function of Travlendar+ is to automatically compute and suggest to the user the best travel among those which fit the preferences he has selected.	Table 1
-	**[F1] Signup and Login**: Travlendar+ users must sign up the first time they intend to create a meeting and further usages of the app will require a login to access all its functionalities.	• Table 7

Table 10: Traceability table

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.
In practice, the low-level components and the functionalities they provide will be developed in this order:

- Sign up/Login manager beans
- Meeting manager bean
- Preferences manager bean
- Route calculator bean
- Conflict checker bean
- Warning manager bean
- Conflict solver bean
- Reminder manager bean

6.2 Macrocomponents to be integrated

The integration test phase for the *Travlendar+* system will be structured based on the architectural division in tiers that is described in [Section 2.2].

With respect to this, the subsystems to be integrated in this phase are the following:

Persistence Tier This includes all the commercial database structures that will be used for the data storage and management of the system, the DBMS will need to be integrated with the Application Logic tier.

Application Logic Tier This includes all the business logic for the application, the data access components and the interfaces components towards external systems and clients. All the interactions among internal logic components must be tested and all the subsystems that interact with this tier must be individually integrated.

Web Tier This includes all the components in charge of the web interface and the communication with the application logic tier and the browser client.

Client Tier This includes the various types of clients, the Mobile Application Client, the Web Browser Client and their internal components. Single clients must behave properly with respect to their internal structure, and must be individually be integrated with the tier they interface with.

6.3 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: $\approxeq$ 30 hours
- Mirko Mantovani working hours: $\approxeq$ 30 hours