POLITECNICO DI MILANO

Software engineering 2 project:

Design Document

version 1.0.0

Travlendar+

Authors:
Giovanni Probo 893698
giovanni.probo@mail.polimi.it

Giovanni Tommasi 900074
giovanni2.tommasi@mail.polimi.it

Academic year 2017-2018

1. INTRODUCTION

1.A - PURPOSE

This document is addressed to the developers and aims to identify the architecture and system design of Travlendar+, a calendar-based mobile application that computes and accounts for travel time between appointments.
In the following sections are described:

- High level architecture
- Main components and their interfaces
- Design of architectural patterns
- Deployment and Test planning

1.B - SCOPE

The system is designed to try to achieve certain properties while providing some functionalities which should help users organizing meetings during the day.

User functionalities:

[G1] - Manage his/her own meetings
[G2] - Move through the city with best mobility options taking in account personal preferences as choice criteria
[G3] - Buy transportation tickets
[G4] - Locate nearest bike from a bike-sharing service
[G5] - Locate nearest car from a car-sharing service
[G6] - Organize lunch breaks
[G7] - Know when get prepared for the journey
[G8] - Know whenever there is a delay in the schedule and know alternative options to reach the meeting in time

Properties

[G1] - Security of sensitive data and confidential information

[G2] - Low latency between the request to the server and its response

[G3] - Deployment and maintenance cheapness

[G4] -Robustness in coping with errors, exceptions and erroneous input.

1.C - DEFINITIONS, ACRONYMS AND ABBREVIATIONS

All the definitions and acronyms listed below represent common words and abbreviations that developers related to a particular meaning in Travlendar+ project context.

1.C.1 - DEFINITIONS

ALTERNATIVE OPTIONS:
Mobility options the system calculates violating user preferences

CALENDAR:
Part of the application showing weekdays and all the event occurring during that period.

CONTROLLER
Software component which is able to modify the model in the database and has some logic functions inside.

CREDENTIALS (USERNAME AND PASSWORD):
Information required in sign up and login operations. Username correspond to the user E-mail while Password is a serie of alphanumeric characters invented by the user whose length is equal or greater than 6 characters.

DAILY PREFERENCES:
Set of settings the user can enable and modify which last until the day ends.
They try to meet user necessities imposed by random external variables (i.e. broken car, less money than expected)

DEFAULT PREFERENCES:
All the possible settings the user can decide the first time he/she logs into the applications. They can be modified in every moment from the settings menu

DEFAULT SETTINGS:
Initial state of settings menu options before user choices.

HELPER
Software component which is not able to modify the model but has some logic functions inside.

HURRY-UP NOTIFICATION:
A notification that is sent from the system to the user, that indicates to be ready for the travel.

INCONSISTENCY
The condition of irregularity in which an option get in conflict with system settings or creates a paradox, an antinomy (15 minutes of travelling, necessity to be at the arrival point in 10).

LATE NOTIFICATION:
A notification that is sent from the system to the user when he/she is late.

LATE OPTIONS
Mobility options the system calculates violating time constraints and user preferences in order to reach the meeting as fast as possible.

MANAGER
Software interface which is able to invoke the right controllers.

MEETING:
Event of flexible duration and placement

REGISTRATION:
Operation the user accomplishes providing his/her e-mail and creating a password. A message will be sent to the user with a proper link to confirm the email address.

SECONDARY OPTIONS:
Not optimal mobility options the system calculates preserving user preferences and time constraints

USER:
Any person who interacts with the application after the registration process.

VISITOR
Any person who interacts with the application before the registration process.

WEATHER FORECAST
Prevision based weather data that combine specific locations with weather categories (i.e sunny, rainy, windy) during a period of time.

1.C.2 - ABBREVIATIONS

3G:
mobile communications standard that allows mobile phones, computers, and other portable electronic devices to access the Internet wirelessly.

API:
a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.

DAT:
Departure advance time.
It is a parameter the user can set to impose the number of minutes
he/she wants to receive a notification before the departure time.

GPS:
Global Positioning System.
It is a radio navigation system that allows land, sea, and airborne users to determine their exact location, velocity, and time 24 hours a day, in all weather conditions, anywhere in the world.

OS:
Operative systems (Android, IOS).

RASD:
Requirements Analysis and Specification Document.

1.D - REVISION HISTORY

- 1.0.0 - Alpha version - 22/11/2017

1.E - REFERENCE DOCUMENTS

- Specification document : Mandatory Project Assignments.pdf

- Theory : Architecture and Design in Practice.pdf
- Theory : Design Part I and II.pdf
- Theory : Verification and Validation.pdf

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

- IEEE Std 1016 tm -2009 Standard for Information Technology-System Design-Software Design Descriptions.

1.F - DOCUMENT STRUCTURE

-Section 2: ARCHITECTURAL DESIGN
This section brings a general overview of the architecture and design of the system showing the main division of the components with the component diagram as well as a device map with the deployment diagram. Runtime views are displayed, interfaces and either architectural or design patterns are presented.

-Section 3: ALGORITHM DESIGN
In this section the most fundamental algorithms for the correct working of the system are explained using Java pseudocode. In particular we focus on the different ways to calculate best mobility option.

-Section 4: USER INTERFACE DESIGN
This section display the main parts the application through mobile and tablet mockups.

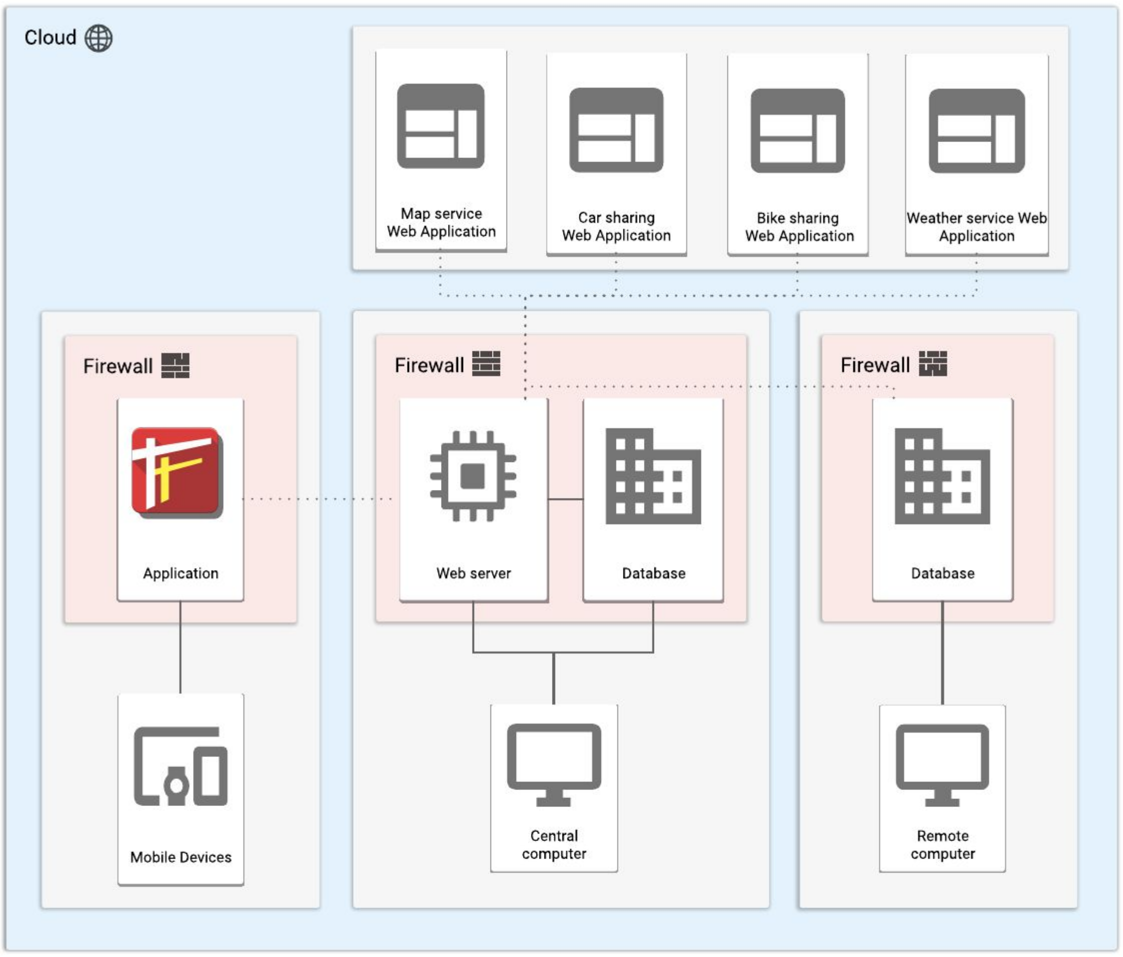
-Section 5 : REQUIREMENTS TRACEABILITY
In this section the most important functional requirements obtained in *Requirement Analysis and Specification Document* are matched to every *Design Document* decision.

-Section 7: IMPLEMENTATION, INTEGRATION AND TEST PLAN
In this section we focused on how we want to plan Travlendar+ implementation, integration and unit tests according to pattern decisions and architectural design.

2. ARCHITECTURAL DESIGN

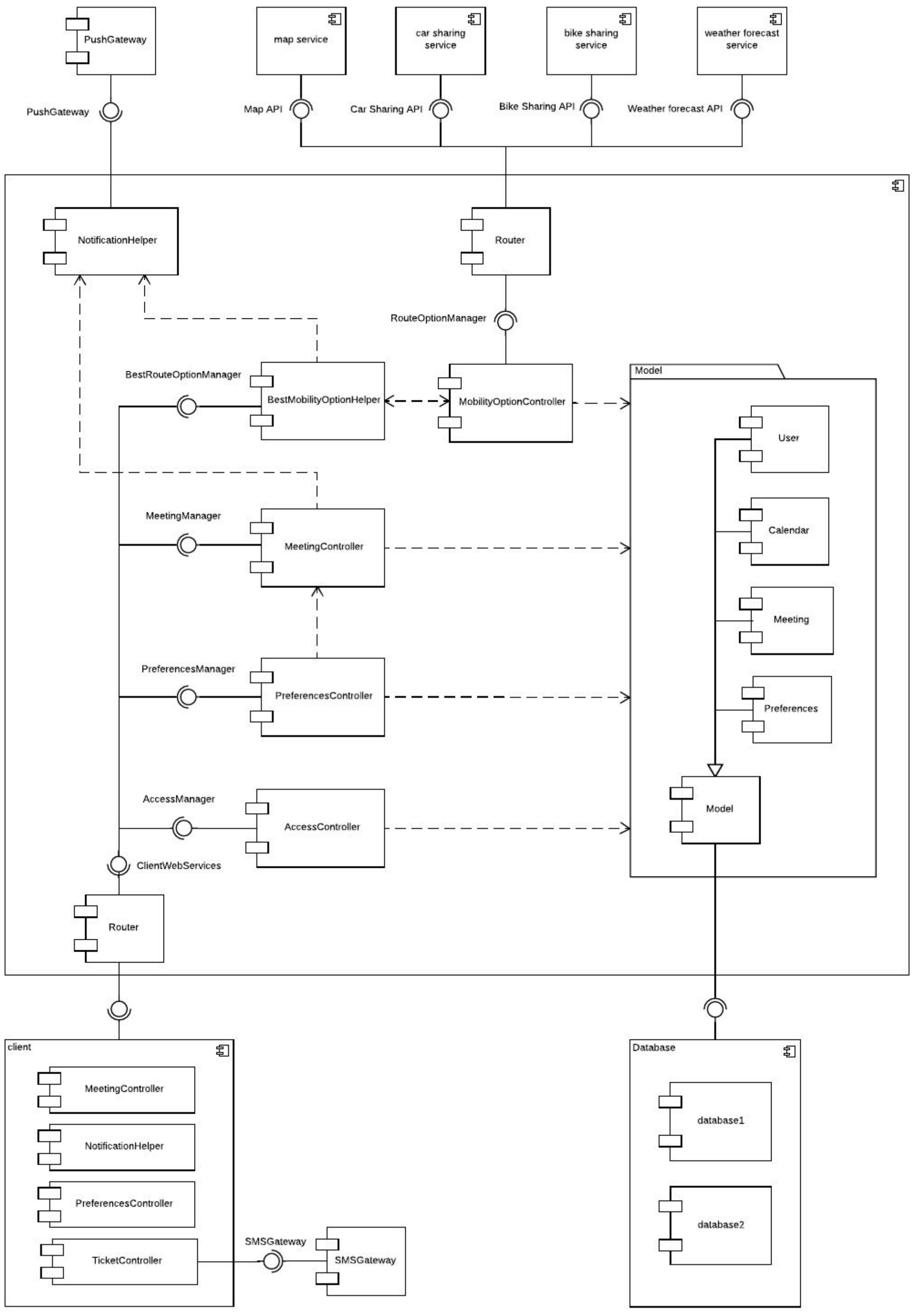
2.A - OVERVIEW

Travlendar+ architecture is composed by three tiers.
It is mainly accessible using a mobile application installed on a portable device. The application handles basic requests and personal data, it is designed to retrieve and display all the information to the users.
The system incorporates also a web server which holds most of the application logic, makes requests to external services and elaborates best route options.
The main server resides in a data center and it is directly linked to a primary database with confidential information.
A secondary database is accessible through a cloud service and contains most of the users data. Here we include all the meetings, users preferences and information for buying tickets.


Cloud
Map service Web Application
Car sharing Web Application
Bike sharing Web Application
Weather service Web Application
Firewall
Application
Mobile Devices
Firewall
Web server
Database
Central computer
Firewall
Database
Remote computer

2.B - COMPONENT VIEW

Component view diagram points out the main components of the system and the interfaces through which they interact.



2.B.1 - CLIENT COMPONENTS

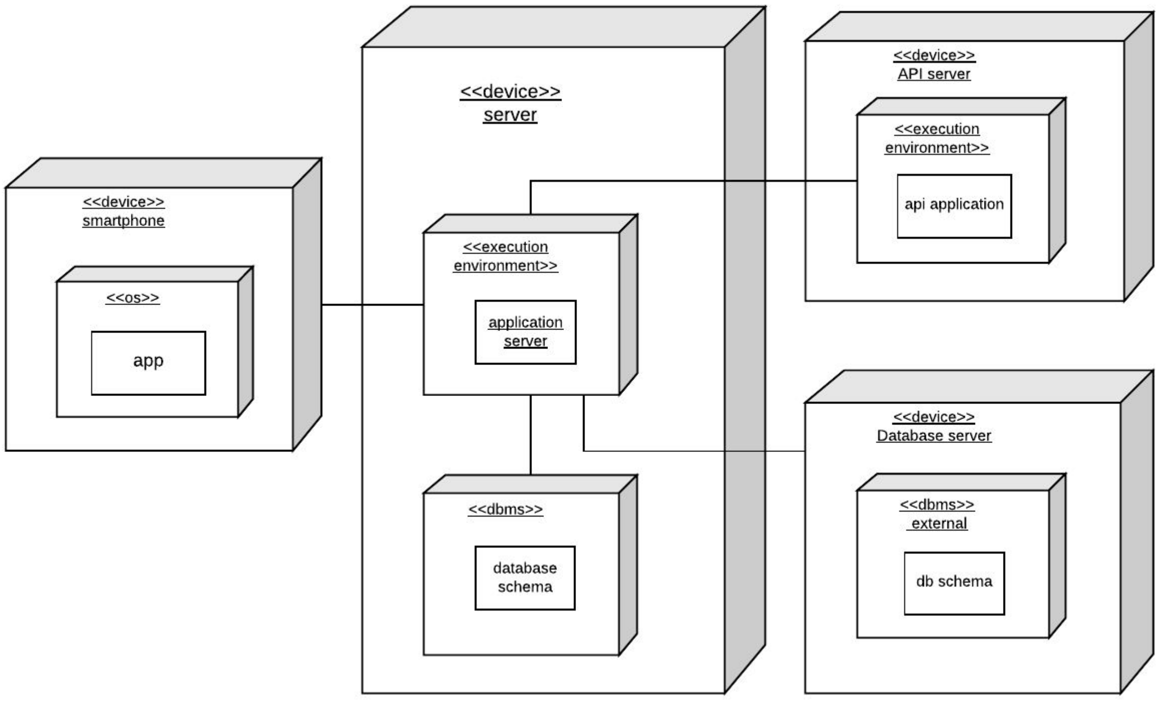
- ACCESS CONTROLLER makes requests to the web server for login and signup operations
- VIEW CONTROLLER display application pages in response to user interactions
- MEETING CONTROLLER check meetings overlap and insert the meeting in the calendar, then send a “Meeting” object to the web server
- PREFERENCE CONTROLLER set the preferences locally then sends a “Preferences” object to the web server
- NOTIFICATION HELPER provides the notification received either by the web server or by the operative system
- TICKET CONTROLLER sends an SMS to the ticket service to buy a ticket with phone credit

2.B.2 - WEB SERVER COMPONENTS

- ACCESS CONTROLLER makes database queries to check user credentials and sends the response to the mobile application
- BEST MOBILITY OPTION HELPER calculate the best mobility option given every possible mobility option
- MOBILITY OPTION CONTROLLER make requests to external services to obtain every possible mobility option
- MEETING CONTROLLER check time, place and preference consistency given a “Meeting” object, then save it to the database
- PREFERENCE CONTROLLER saves in the database user preferences given a “Preference” object
- NOTIFICATION HELPER sends notifications to the mobile application when the system need to notify an update

2.C - DEPLOYMENT VIEW

Deployment view diagram points out the hardware configuration of the overall system architecture and which software component run on each node.



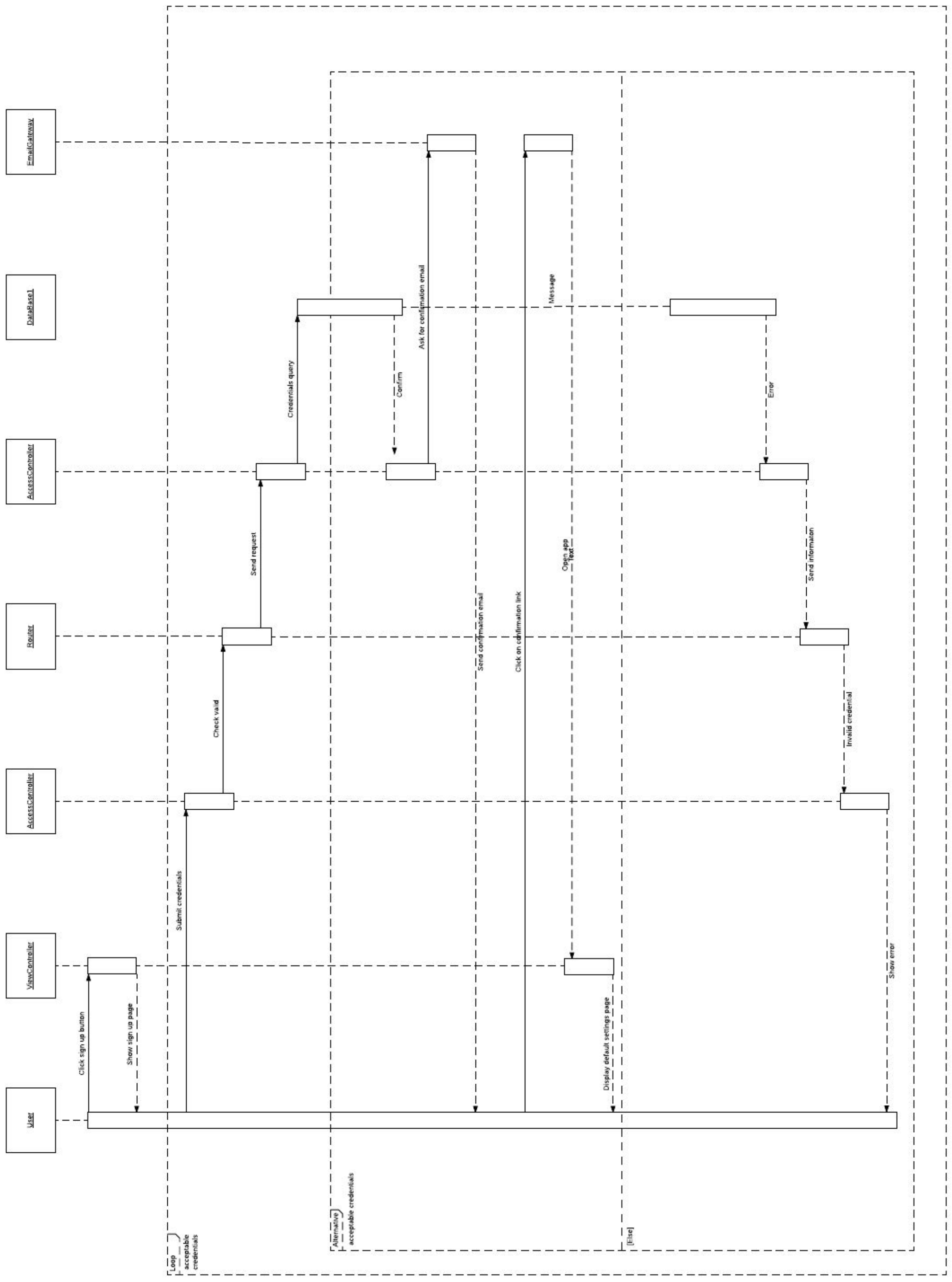
The system architecture is composed by a mobile application, a web server and two databases, which form in total 3 tiers. In the diagram also the external services are represented to completely understand the overall architecture.

- CLIENT TIER : it is composed by a mobile application running on Android or IOS operating system which is respectively coded in Java and Swift languages.
- APPLICATION TIER: it is composed by a web server application coded in Java EE running on a server computer (i.e. Dell PowerEdge T20 [Xeon]).
- DATA TIER: It is composed by two relational databases. The first one is directly linked to the web server and running on the same machine as It contain the most sensitive data. The second one is accessible via cloud and contains any other user information.
- EXTERNAL SERVICES: Web applications accessible via their own Apis. Most likely these services are runned on remote computers and transfer data using network connection.

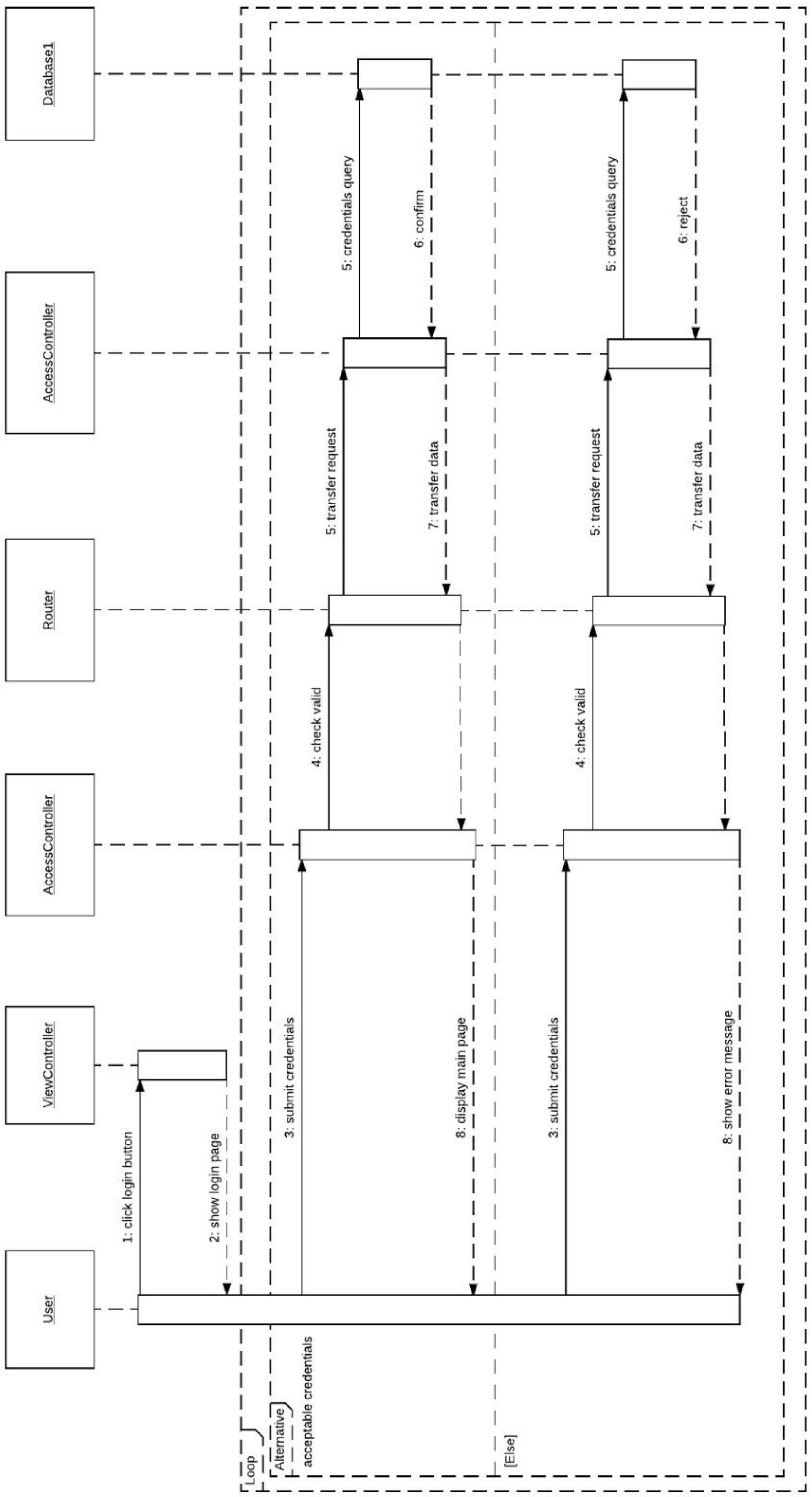
2.D - RUNTIME VIEW

The following Runtime views concern the most important activity of Travlendar+.

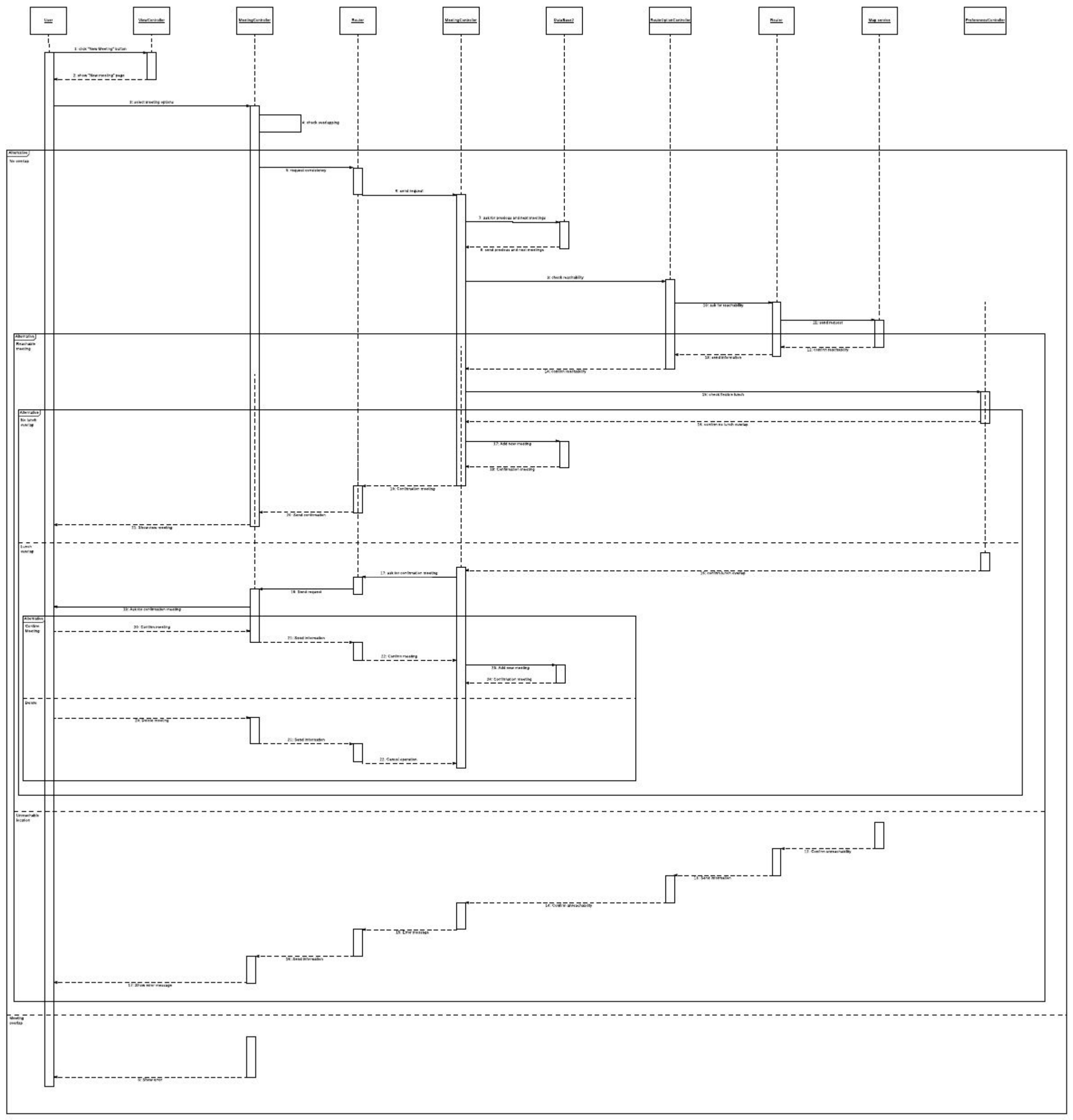
2.D.1 - REGISTRATION PROCESS



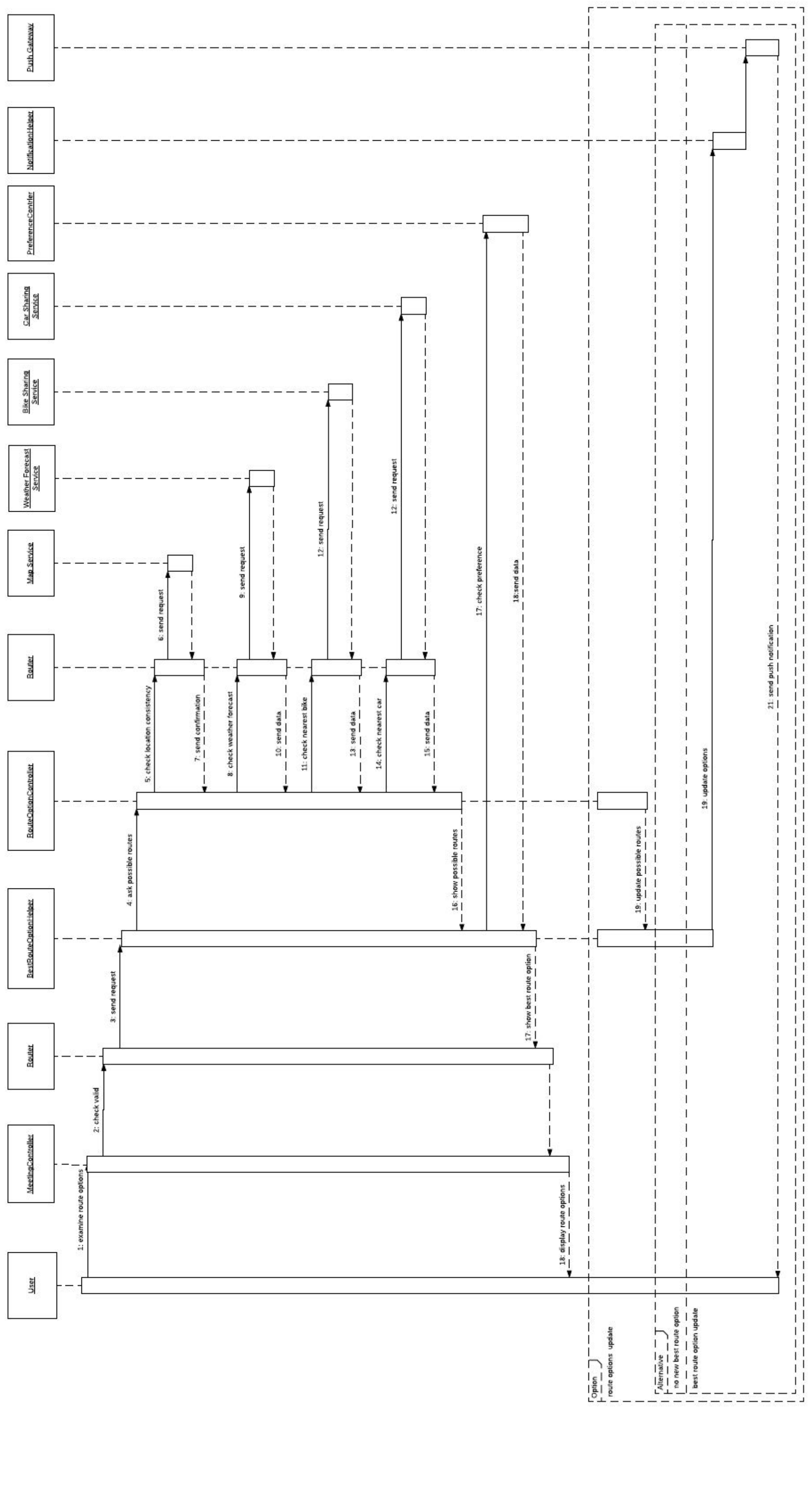
2.D.2 - LOGIN


User
ViewController
AccessController
Router
AccessController
Database1
Loop
Alternative
acceptable credentials
[Else]
1: click login button
2: show login page
3: submit credentials
4: check valid
5: transfer request
5: credentials query
6: confirm
7: transfer data
8: display main page
3: submit credentials
4: check valid
5: transfer request
5: credentials query
6: reject
7: transfer data
8: show error message

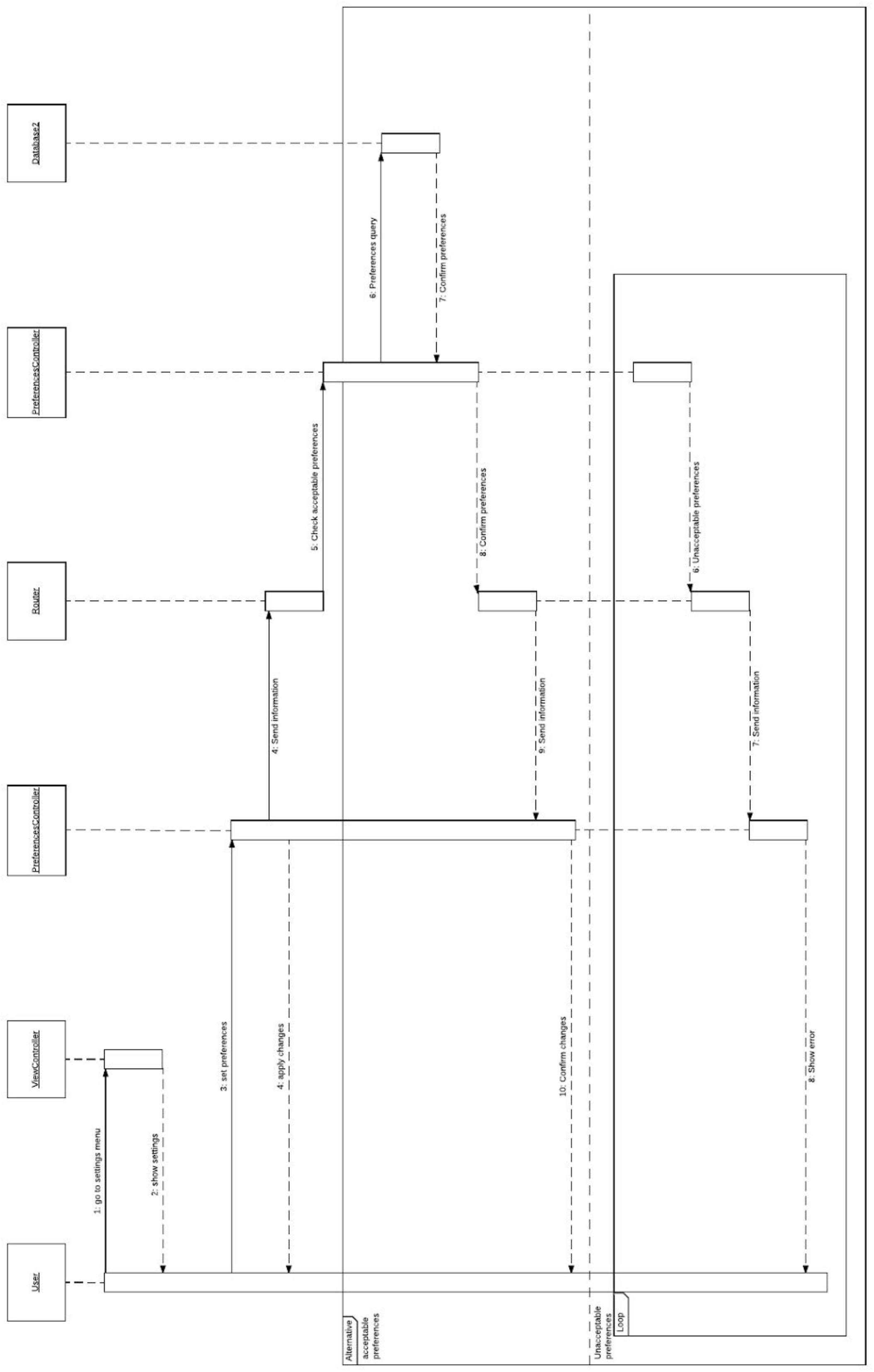
2.D.3 - CREATION OF A MEETING



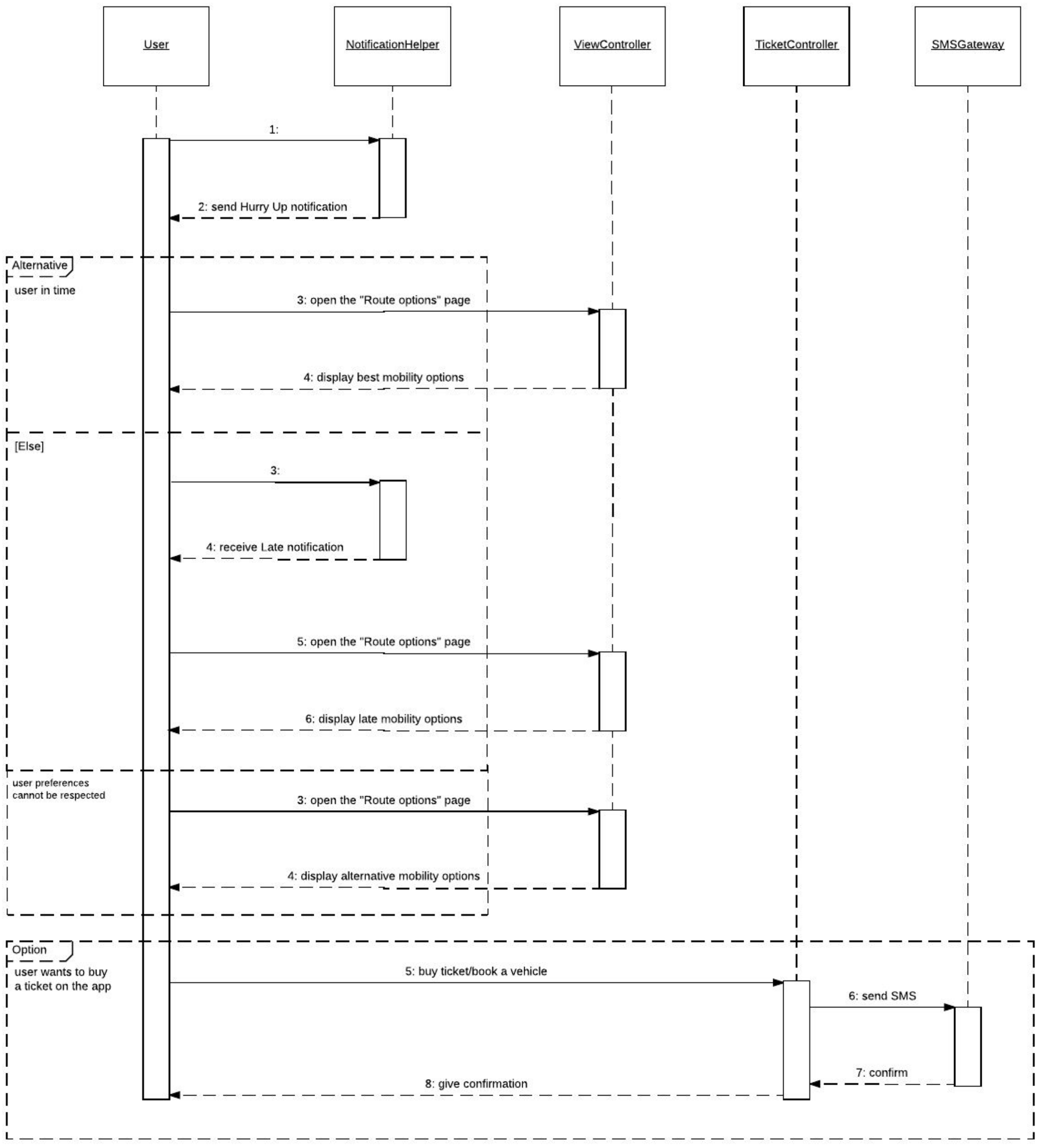
2.D.4 - EXAMINE BEST MOBILITY OPTIONS



2.D.5 - SETTING PREFERENCES



2.D.6 - STARTING A JOURNEY


User
NotificationHelper
ViewController
TicketController
SMSGateway
1:
2: send Hurry Up notification
Alternative
user in time
3: open the "Route options" page
4: display best mobility options
[Else]
3:
4: receive Late notification
5: open the "Route options" page
6: display late mobility options
user preferences cannot be respected
3: open the "Route options" page
4: display alternative mobility options
Option
user wants to buy a ticket on the app
5: buy ticket/book a vehicle
6: send SMS
7: confirm
8: give confirmation

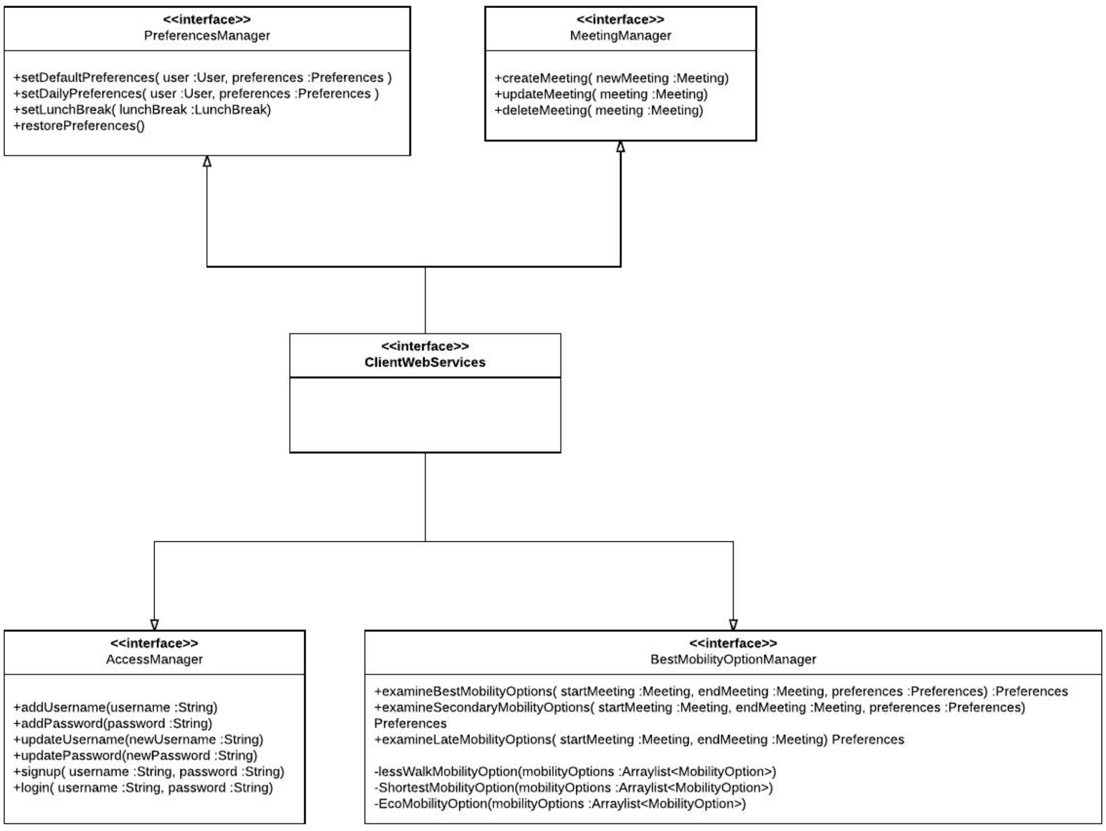
2.F - COMPONENT INTERFACES

Travlendar+ system is composed by software component interfaces called managers which handle the system information flow, redirecting the data and invoking the right controllers.

2.F.1 - WEB SERVER INTERFACES

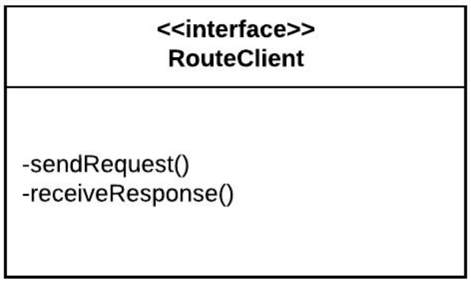
ClientWebServices interface gathers all the methods required to perform a client request.
It is subdivided in :

- AccessManager collects all the methods to perform a registration or a login in the system.
- BestMobilityOptionManager collects all the methods to send to a client best mobility options.
- MeetingManager collects all the methods to manage meeting creation according to time place and preference consistency, meeting update and deletion.
- PreferenceManager collects all the methods to change some user preferences.


<<interface>>
PreferencesManager
+setDefaultPreferences(user :User, preferences :Preferences)
+setDailyPreferences(user :User, preferences :Preferences)
+setLunchBreak(lunchBreak :LunchBreak)
+restorePreferences()
<<interface>>
MeetingManager
+createMeeting(newMeeting :Meeting)
+updateMeeting(meeting :Meeting)
+deleteMeeting(meeting :Meeting)
<<interface>>
ClientWebServices
<<interface>>
AccessManager
+addUsername(username :String)
+addPassword(password :String)
+updateUsername(newUsername :String)
+updatePassword(newPassword :String)
+signup(username :String, password :String)
+login(username :String, password :String)
<<interface>>
BestMobilityOptionManager
+examineBestMobilityOptions(startMeeting :Meeting, endMeeting :Meeting, preferences :Preferences) :Preferences
+examineSecondaryMobilityOptions(startMeeting :Meeting, endMeeting :Meeting, preferences :Preferences) Preferences
+examineLateMobilityOptions(startMeeting :Meeting, endMeeting :Meeting) Preferences
-lessWalkMobilityOption(mobilityOptions :Arraylist<MobilityOption>)
-ShortestMobilityOption(mobilityOptions :Arraylist<MobilityOption>)
-EcoMobilityOption(mobilityOptions :Arraylist<MobilityOption>)

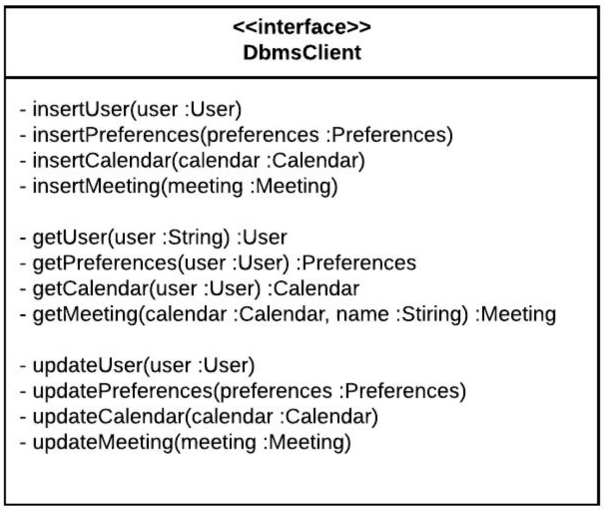
2.F.2 - ROUTER INTERFACE

RouterClient interface gathers the methods to send and receive data between the different part of the system.



2.F.3 - DATABASE INTERFACE

DbmsClient gathers all the methods to make queries to both the databases.



<<interface>>
DbmsClient
- insertUser(user :User)
- insertPreferences(preferences :Preferences)
- insertCalendar(calendar :Calendar)
- insertMeeting(meeting :Meeting)
- getUser(user :String) :User
- getPreferences(user :User) :Preferences
- getCalendar(user :User) :Calendar
- getMeeting(calendar :Calendar, name :Stiring) :Meeting
- updateUser(user :User)
- updatePreferences(preferences :Preferences)
- updateCalendar(calendar :Calendar)
- updateMeeting(meeting :Meeting)

2.G - SELECTED ARCHITECTURAL PATTERNS AND PATTERNS

2.G.1 - ARCHITECTURAL PATTERNS

2.G.1.1 - MODEL VIEW CONTROLLER

It divides the application into three interconnected parts:

- The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface.
- The view can be any representation of information, it generates new output to the user based on changes in the model.
- The controller accepts input and converts it to commands for the model or view.

MVC design pattern decouples these major components separating internal representations of information from the ways information is presented to the user.

2.G.1.2 - MULTITIER ARCHITECTURE

It is a client–server architecture in which presentation, application processing, and data management functions are physically separated.

- The presentation tier displays information directly to the user.
- The application tier controls the functionality of the application by performing detailed processing.
- The data tier includes the data persistence mechanisms and the data access layer that encapsulates the persistence mechanisms and exposes the data.\

In Travlendar+ app these are represented by the mobile application, the web server and the database.

2.G.1.3 - EVENT DRIVER ARCHITECTURE

This architectural pattern promotes the production, detection, consumption of, and reaction to events. The information is spread through the application using messages called event notifications, and not the event itself, which is the state change that triggers the message emission.

Travlendar+ logic reacts to events that originates from the mobile application and the external services answering with proper information.

2.G.1.4 - ONLINE TRANSACTION PROCESSING ARCHITECTURE

Architectural pattern that characterize the data storage organization in order to be accessed by a large number of short on-line transactions.

The key goals of OLTP applications are availability, speed, concurrency and recoverability.

Travlendar+ use two transactional databases in order to achieve this behaviour.

2.G.2 - SOFTWARE DESIGN PATTERNS

2.G.2.1 - STRATEGY PATTERN

Behavioral software design pattern that enables selecting an algorithm at runtime. The strategy pattern defines a family of algorithms, encapsulates each algorithm, and makes the algorithms interchangeable within that family.

In Travlendar+ mobility options can be of three types: best mobility options, secondary mobility options and late mobility options.

Each one has a specific algorithm which is executed at runtime taking in account user preferences and time constrictions in different ways.

The strategy pattern helps the system switching among this algorithms when necessary.

2.G.2.2 - ADAPTER PATTERN

Software pattern that allows the interface of an existing class to be used as another interface.

In Travlendar+ external services Api are adapted to be used with component interfaces.

2.G.2.3 - OBSERVER PATTERN

Software design pattern in which an object maintains a list of its dependents, called observers, and notifies them automatically of any state changes calling one of their methods.

In Travlendar+ it is included in the application of the MVC architectural pattern.

2.G.2.4 - PROXY PATTERN

Software design pattern in which a class is functioning as an interface to something else.

In Travlendar+ the proxy pattern is used either when a client interfaces with the web server using the network connection or during an access to the database.

This let the system have more control when accessing a sensitive object from the mobile application and add some error debugging functionalities.

2.H - OTHER DESIGN DECISIONS

We decided to use two different databases:

- The first database is a local wired HDD, which contains sensitive information (i.e. Username and Password)
- The second database is an On-Cloud Database, which contains all non-sensitive data

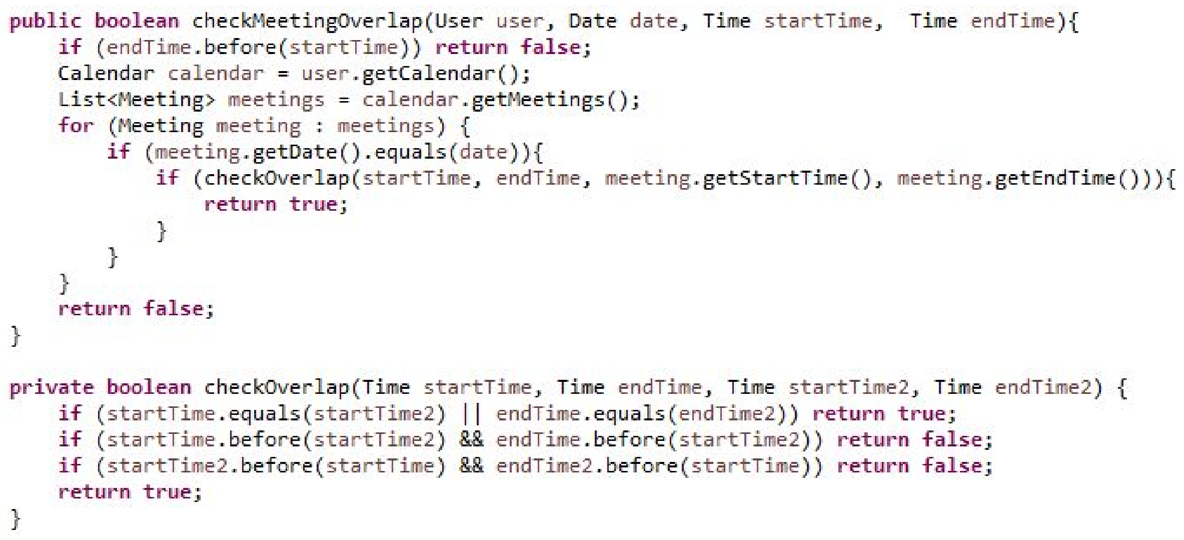
This choice is due to the will to protect our users' informations and to the will to reduce unnecessary cost.

3. ALGORITHM DESIGN

The following are the most critical algorithm used by Travlendar+

3.A - MEETING OVERLAP ALGORITHM

The following algorithm is used by the user's smartphone to check whether a meeting overlap occur. If checkMeetingOverlap return false, the creation of the meeting goes on and the mobile app will send a request of meeting creation to the central server.



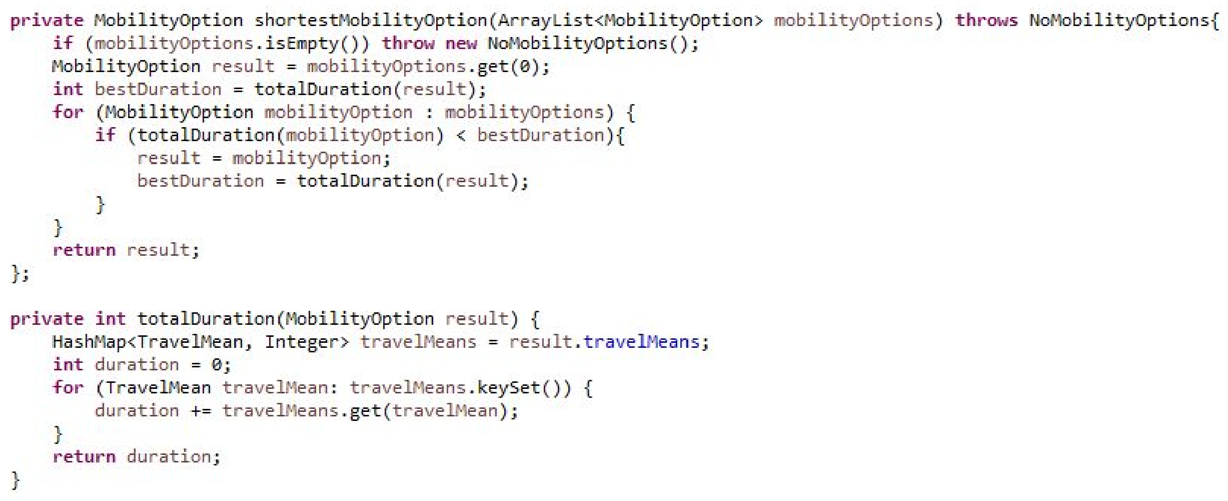
```java
public boolean checkMeetingOverlap(User user, Date date, Time startTime,  Time endTime){
    if (endTime.before(startTime)) return false;
    Calendar calendar = user.getCalendar();
    List<Meeting> meetings = calendar.getMeetings();
    for (Meeting meeting : meetings) {
        if (meeting.getDate().equals(date)){
            if (checkOverlap(startTime, endTime, meeting.getStartTime(), meeting.getEndTime())){
                return true;
            }
        }
    }
    return false;
}

private boolean checkOverlap(Time startTime, Time endTime, Time startTime2, Time endTime2) {
    if (startTime.equals(startTime2) || endTime.equals(endTime2)) return true;
    if (startTime.before(startTime2) && endTime.before(startTime2)) return false;
    if (startTime2.before(startTime) && endTime2.before(startTime)) return false;
    return true;
}
```

3.B - BEST MOBILITY OPTION ALGORITHMS

The following algorithms describes three way to find the best mobility option, according to the user’s preferences.

3.B.1 - SHORTEST MOBILITY OPTION ALGORITHM

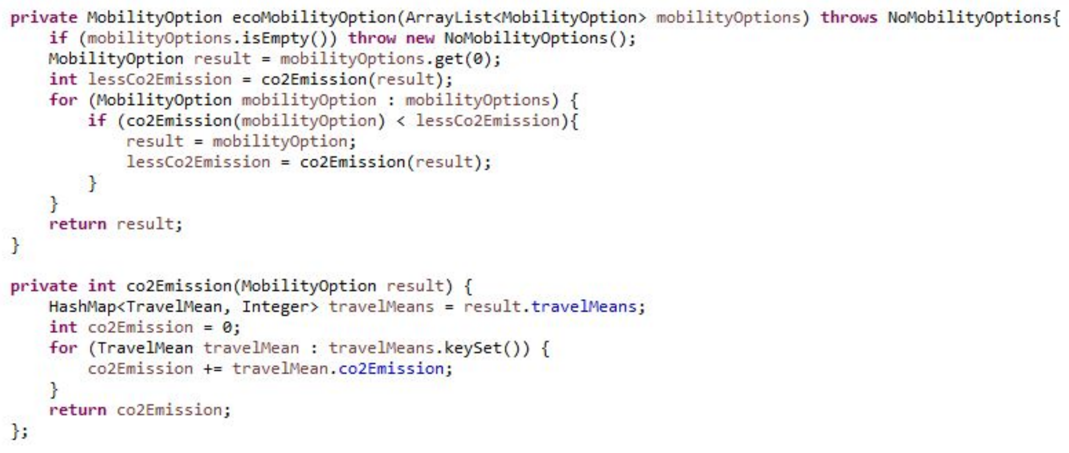
The first algorithm is one of the most common. The algorithm find the shortest mobility option, i.e. the mobility option that spend the shortest possible time.



```java
private MobilityOption shortestMobilityOption(ArrayList<MobilityOption> mobilityOptions) throws NoMobilityOptions{
    if (mobilityOptions.isEmpty()) throw new NoMobilityOptions();
    MobilityOption result = mobilityOptions.get(0);
    int bestDuration = totalDuration(result);
    for (MobilityOption mobilityOption : mobilityOptions) {
        if (totalDuration(mobilityOption) < bestDuration){
            result = mobilityOption;
            bestDuration = totalDuration(result);
        }
    }
    return result;
};

private int totalDuration(MobilityOption result) {
    HashMap<TravelMean, Integer> travelMeans = result.travelMeans;
    int duration = 0;
    for (TravelMean travelMean: travelMeans.keySet()) {
        duration += travelMeans.get(travelMean);
    }
    return duration;
}
```

3.B.2 - ECO MOBILITY OPTION ALGORITHM

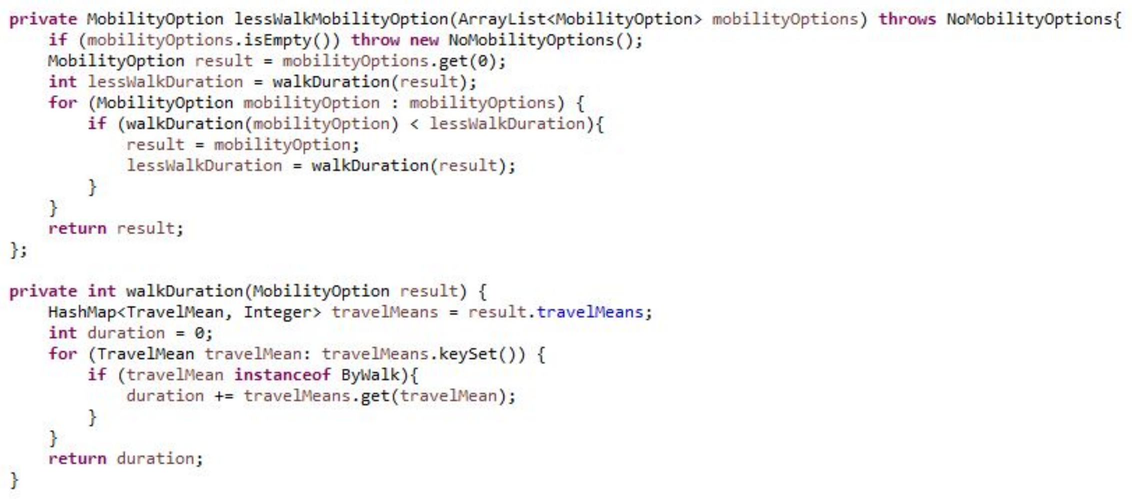
This algorithm find the mobility option with the lowest CO_2 emission.



```java
private MobilityOption ecoMobilityOption(ArrayList<MobilityOption> mobilityOptions) throws NoMobilityOptions{
    if (mobilityOptions.isEmpty()) throw new NoMobilityOptions();
    MobilityOption result = mobilityOptions.get(0);
    int lessCo2Emission = co2Emission(result);
    for (MobilityOption mobilityOption : mobilityOptions) {
        if (co2Emission(mobilityOption) < lessCo2Emission){
            result = mobilityOption;
            lessCo2Emission = co2Emission(result);
        }
    }
    return result;
}

private int co2Emission(MobilityOption result) {
    HashMap<TravelMean, Integer> travelMeans = result.travelMeans;
    int co2Emission = 0;
    for (TravelMean travelMean : travelMeans.keySet()) {
        co2Emission += travelMean.co2Emission;
    }
    return co2Emission;
};
```

3.B.3 - SHORTEST TIME BY WALK MOBILITY OPTION ALGORITHM

The following algorithm find the mobility option with the shortest time by walk.

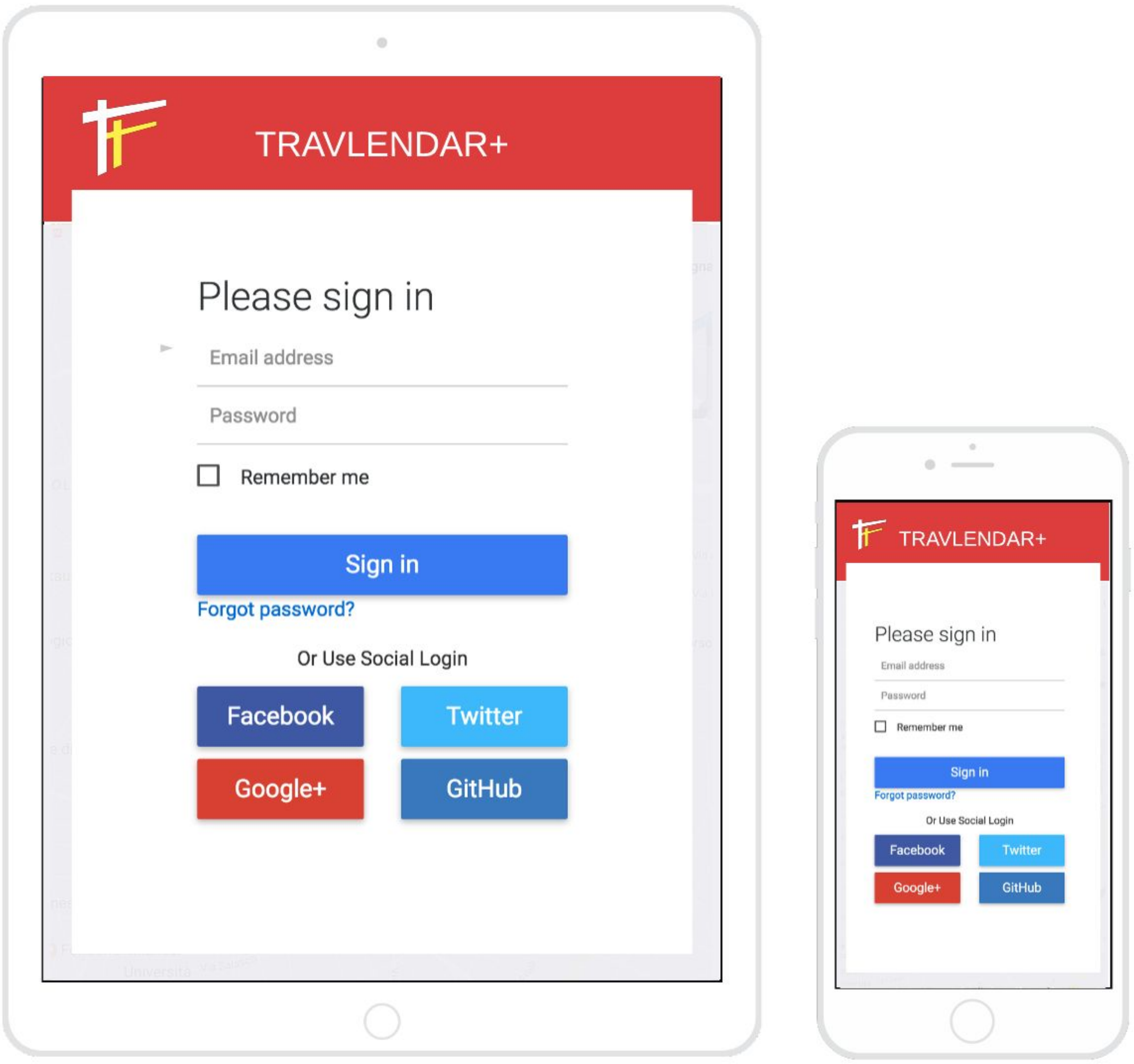


```java
private MobilityOption lessWalkMobilityOption(ArrayList<MobilityOption> mobilityOptions) throws NoMobilityOptions{
    if (mobilityOptions.isEmpty()) throw new NoMobilityOptions();
    MobilityOption result = mobilityOptions.get(0);
    int lessWalkDuration = walkDuration(result);
    for (MobilityOption mobilityOption : mobilityOptions) {
        if (walkDuration(mobilityOption) < lessWalkDuration){
            result = mobilityOption;
            lessWalkDuration = walkDuration(result);
        }
    }
    return result;
};

private int walkDuration(MobilityOption result) {
    HashMap<TravelMean, Integer> travelMeans = result.travelMeans;
    int duration = 0;
    for (TravelMean travelMean: travelMeans.keySet()) {
        if (travelMean instanceof ByWalk){
            duration += travelMeans.get(travelMean);
        }
    }
    return duration;
}
```

4. USER INTERFACE DESIGN

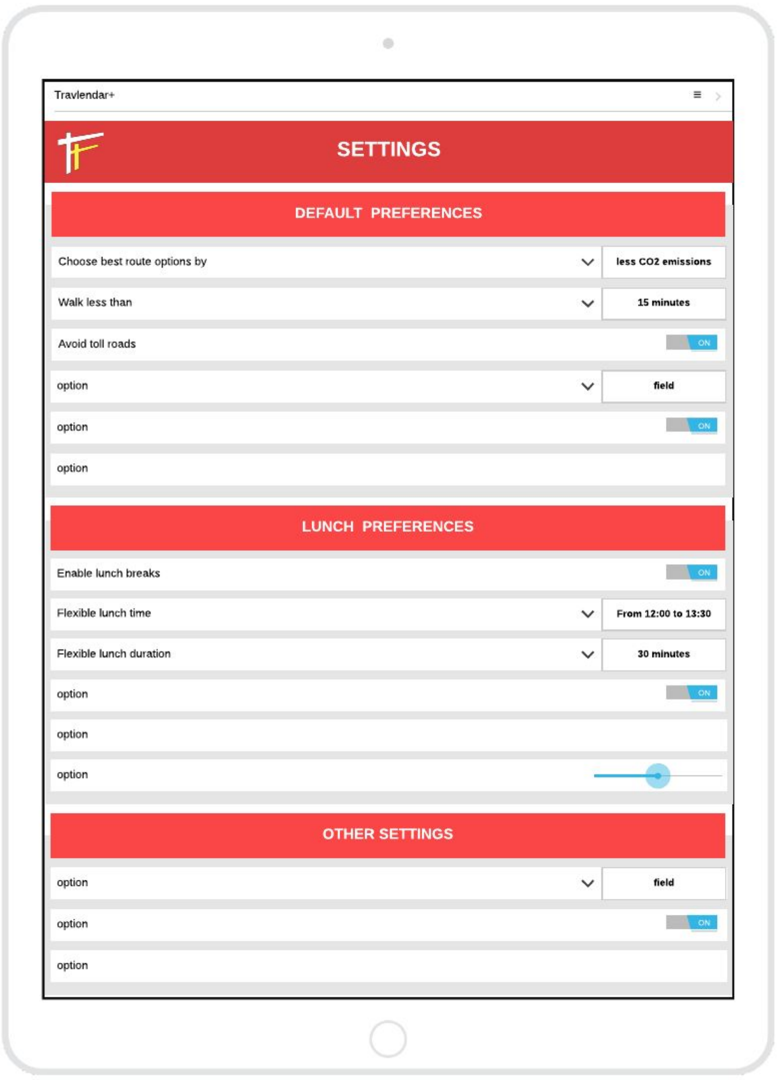
4.A - LOGIN AND SIGN UP

This is the first page that will be displayed by the user. Here it is possible to log into the system by inserting credentials, or to sign up and provide e-mail, Username and Password.



4.B - DEFAULT PREFERENCES

When a new user sign up to Travlendar+, he/she should be able to set the his/her default preferences or accept the recommended ones.
This page can also be reached using ≡ button whenever the user desires to change one of the preferences.


Travlendar+ ≡

SETTINGS

DEFAULT PREFERENCES

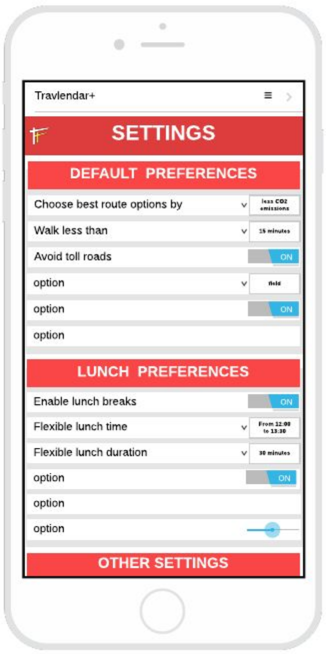
Choose best route options by	less CO2 emissions
Walk less than	15 minutes
Avoid toll roads	ON
option	field
option	ON
option	

LUNCH PREFERENCES

Enable lunch breaks	ON
Flexible lunch time	From 12:00 to 13:30
Flexible lunch duration	30 minutes
option	ON
option	
option	

OTHER SETTINGS

option	field
option	ON
option	


Travlendar+ ≡

SETTINGS

DEFAULT PREFERENCES

Choose best route options by	less CO2 emissions
Walk less than	15 minutes
Avoid toll roads	ON
option	field
option	ON
option	

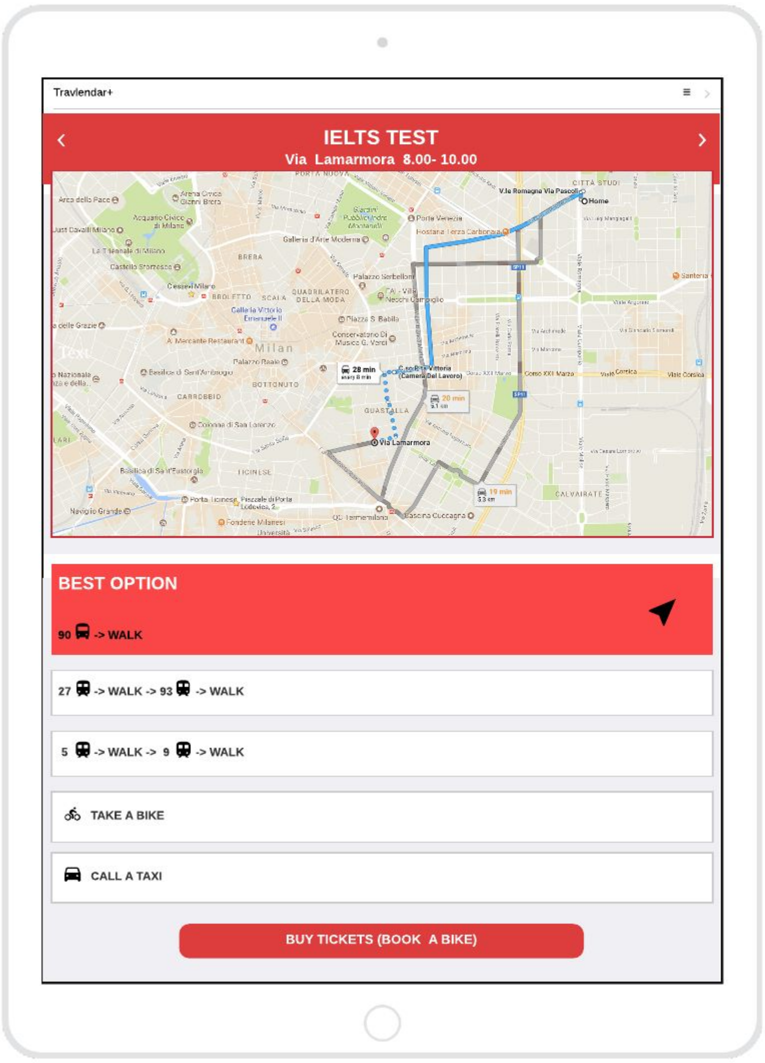
LUNCH PREFERENCES

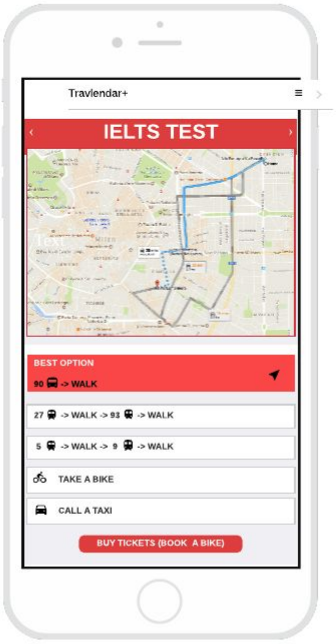
Enable lunch breaks	ON
Flexible lunch time	From 12:00 to 13:30
Flexible lunch duration	30 minutes
option	ON
option	
option	

OTHER SETTINGS

4.C - MAIN PAGE

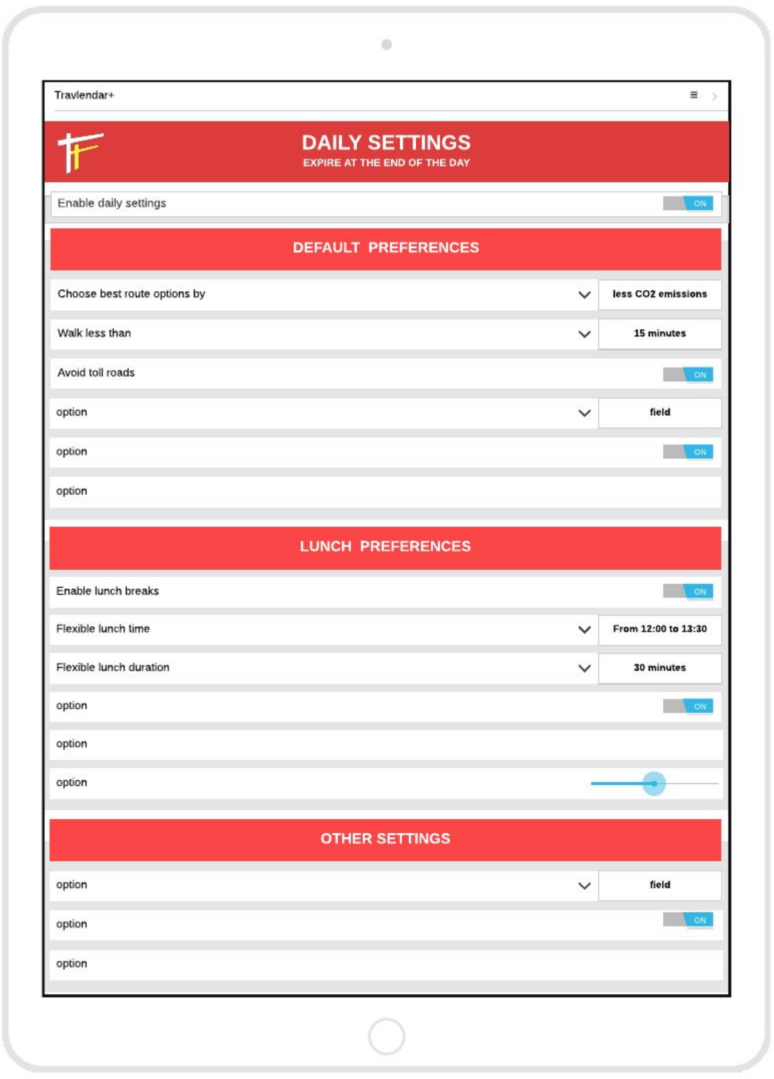
This page will be shown once logged in. It contains all the incoming meetings and each best mobility option. The first incoming meeting will show a secondary mobility option and the lunch break will appear if set.
Clicking on a meeting “Suggested routes” page will be opened.
On the bottom of the page “Change daily settings” button will redirect the user to Daily Settings page.
On the top right ≡ button will let the user moving through the pages (Main, Meetings, Settings).





4.D - DAILY SETTINGS

This page is quite similar to Default Preferences page. In this page user can modify the preferences for the current day because of unexpected events (i.e. broken car, no money to buy transport tickets etc.).



Travlendar+

DAILY SETTINGS
EXPIRE AT THE END OF THE DAY

Enable daily settings	ON

DEFAULT PREFERENCES

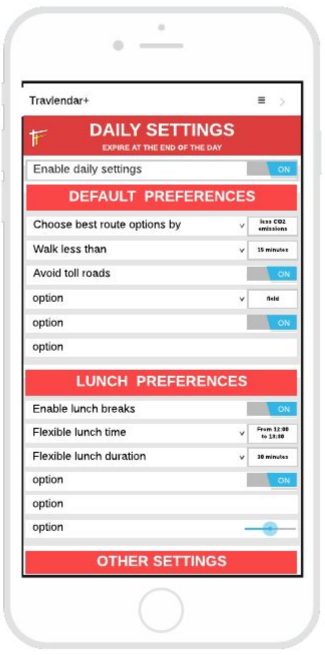
Choose best route options by	less CO2 emissions
Walk less than	15 minutes
Avoid toll roads	ON
option	field
option	ON
option	

LUNCH PREFERENCES

Enable lunch breaks	ON
Flexible lunch time	From 12:00 to 13:30
Flexible lunch duration	30 minutes
option	ON
option	
option	

OTHER SETTINGS

option	field
option	ON
option	



Travlendar+

DAILY SETTINGS
EXPIRE AT THE END OF THE DAY

Enable daily settings	ON

DEFAULT PREFERENCES

Choose best route options by	less CO2 emissions
Walk less than	15 minutes
Avoid toll roads	ON
option	field
option	ON
option	

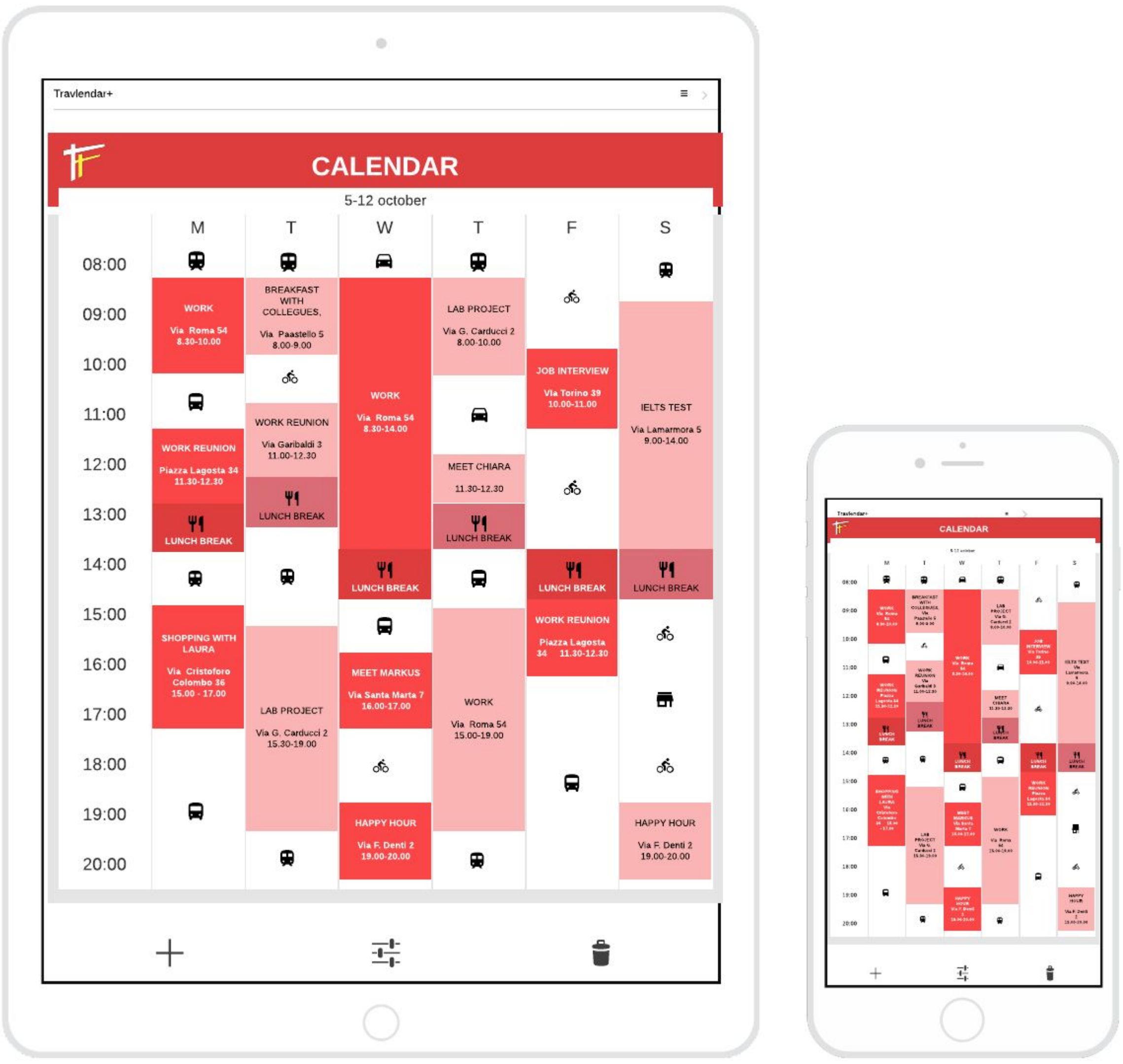
LUNCH PREFERENCES

Enable lunch breaks	ON
Flexible lunch time	From 12:00 to 13:30
Flexible lunch duration	30 minutes
option	ON
option	
option	

OTHER SETTINGS

4.E - CALENDAR

Calendar page shows a weekly calendar with all the created meetings and some details.
On the bottom three button could let the user to create, modify or delete meetings.
Clicking on a meeting “Suggested routes” page will be shown.
Using “Create” or “Modify” buttons user will be redirected to “Meeting details” page.


Travlendar+

CALENDAR

5-12 october

M:
- WORK Via Roma 54 8.30-10.00
- WORK REUNION Piazza Lagosta 34 11.30-12.30
- LUNCH BREAK
- SHOPPING WITH LAURA Via Cristoforo Colombo 36 15.00 - 17.00

T:
- BREAKFAST WITH COLLEGUES, Via Paastello 5 8.00-9.00
- WORK REUNION Via Garibaldi 3 11.00-12.30
- LUNCH BREAK
- LAB PROJECT Via G. Carducci 2 15.30-19.00

W:
- WORK Via Roma 54 8.30-14.00
- LUNCH BREAK
- MEET MARKUS Via Santa Marta 7 16.00-17.00
- HAPPY HOUR Via F. Denti 2 19.00-20.00

T:
- LAB PROJECT Via G. Carducci 2 8.00-10.00
- MEET CHIARA 11.30-12.30
- LUNCH BREAK
- WORK Via Roma 54 15.00-19.00

F:
- JOB INTERVIEW Via Torino 39 10.00-11.00
- LUNCH BREAK
- WORK REUNION Piazza Lagosta 34 11.30-12.30

S:
- IELTS TEST Via Lamarmora 5 9.00-14.00
- LUNCH BREAK
- HAPPY HOUR Via F. Denti 2 19.00-20.00

5. REQUIREMENTS TRACEABILITY

Every design and architectural decision described in this document aims to achieve the goals of Travlendar+ expressed in the *Requirements Analysis and Specification Document*.
Here we include a list of the requirements provided in RASD document and the software components whose intent is cover those specific functionalities.

- In the first page users will be able to log in and sign up to the service.
    - ClientWebServices
        - AccessManager

- During registration users will set default preferences.
    - ClientWebServices
        - AccessManager
        - PreferencesManager
    - DbmsClient

- Users will be able to create, update and delete meetings.
    - ClientWebServices
        - MeetingManager
    - DbmsClient

- During meeting creation users will provide time and location information
    - ClientWebService
        - MeetingManager.

- In “Calendar” page users will be able to visualize all their meetings chronologically ordered and divided by week
    - DbmsClient
    - MeetingController.

- In “Settings” page users will be able to change default preferences or to restore default settings
    - ClientWebServices
        - PreferencesManager
    - DbmsClient.

- In “Settings” page users will find “Lunch break” options to set the interval of time for a lunch break.
    - ClientWebServices
        - PreferencesManager

- In the main page users will find “Change daily settings” button to set daily preferences
    - ClientWebServices
        - PreferencesManager

- The system will take in account preferences to create best mobility options and locating nearest bikes or cars from vehicle-sharing systems.
    - ClientWebServices
        - PreferencesManager
        - MobilityOptionController
        - BestMobilityOptionManager

- In the main page the system will show next meetings and best mobility options for each travel.
    - ClientWebServices
        - BestMobilityOptionManager

- The system will provide alternative mobility options in case of strikes, transports delays and bad weather. The user will be warned with a push notification too.
    - External APIs
    - PushGateway
    - ClientWebServices
        - BestMobilityOptionManager

- Clicking on a meeting the system will display secondary mobility options, a small map and a section for buying tickets or booking bikes and cars.
    - ClientWebServices
        - MobilityOptionController
    - External APIs
    - SMSGateway

- The system will send a “Hurry Up” notification for departure according to DAT.
    - PushGateway

- The system will send a “Late” notification when it’s not possible to reach the location in time anymore.
    - PushGateway

- The system will warn the user with an error message whenever creating a meeting there will be the impossibility to reach the location in time.
    - ClientWebService
    - PushGateway

- The system will warn the user with an error message whenever a meeting will overlap lunch break time.
    - ClientWebService
    - PushGateway

6. IMPLEMENTATION, INTEGRATION AND TEST PLAN

The implementation integration and testing schedule is based on the patterns that we used during this project, i.e. MVC and Multitier architecture.
We have chosen a bottom-up Integration Testing, this allows a more efficient unit testing and it fits better to the chosen patterns.
First, we have to develop and test the Web server model and the basic structure of our Database (Figure 1), using Java EE and MySQL.
After that, Web server controllers will be implemented and tested (Figure 2).
Then, we will add Web server Managers and we will create queries to access the Database and we will check the right behavior of the external services API (Figure 3).
Consequently, we will proceed with the creation of the Client classes that allows some offline features (i.e. Meeting creation and Preferences) (Figure 4). Client side will be developed using Android Studio and XCode.
After that, we will complete Client Controllers, testing if the client is able to send request to the Web server (Figure 5).
Once we have completed the model of the Client side, we will develop the GUI.
Eventually, push notification gates and SMS gates will be implemented and tested (Figure 6).

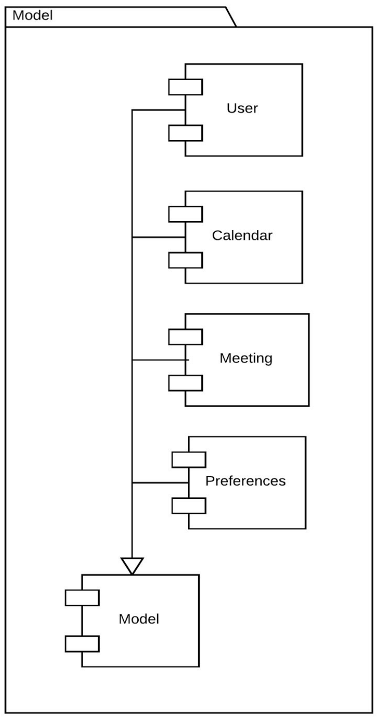


Figure 1

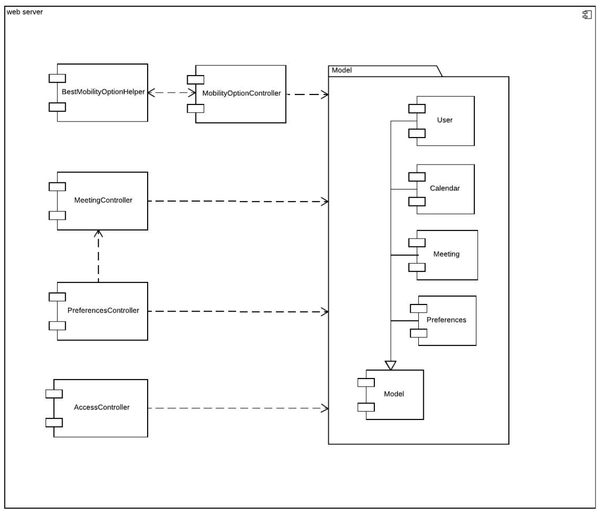


Figure 2

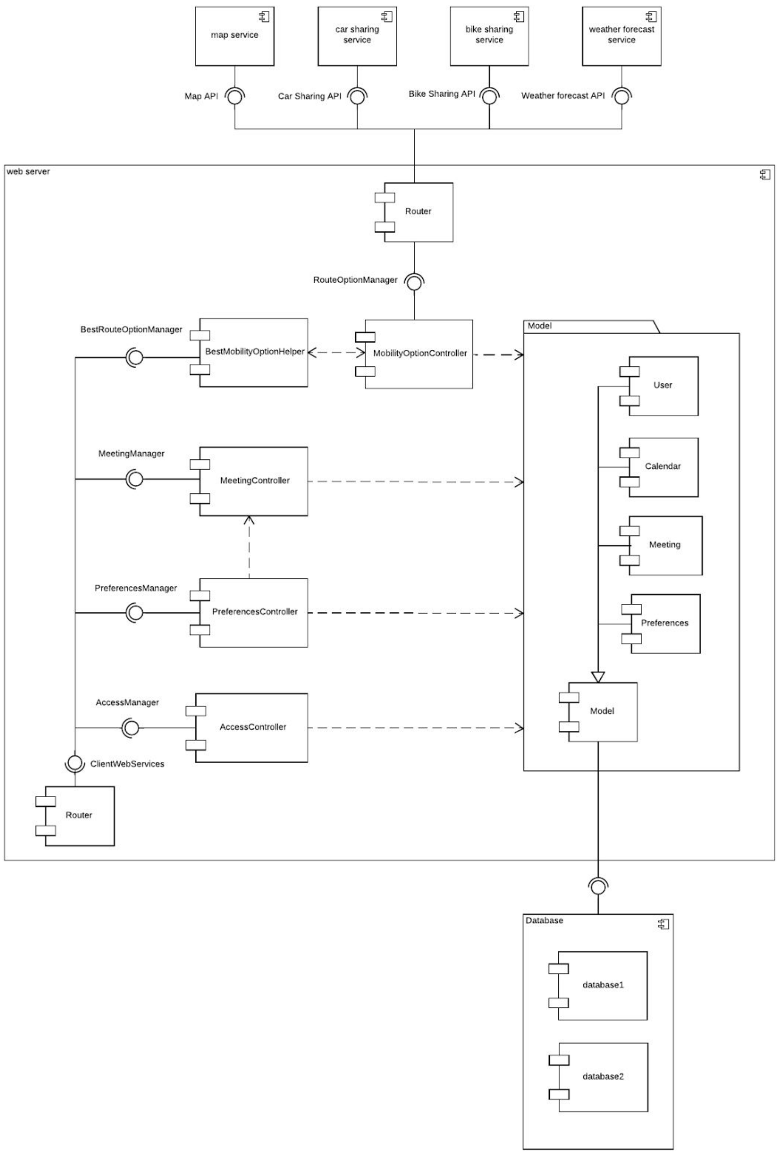


Figure 3

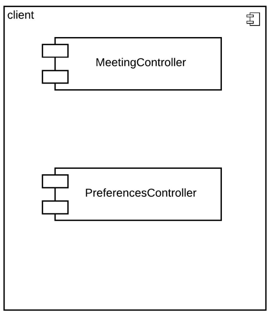


Figure 4

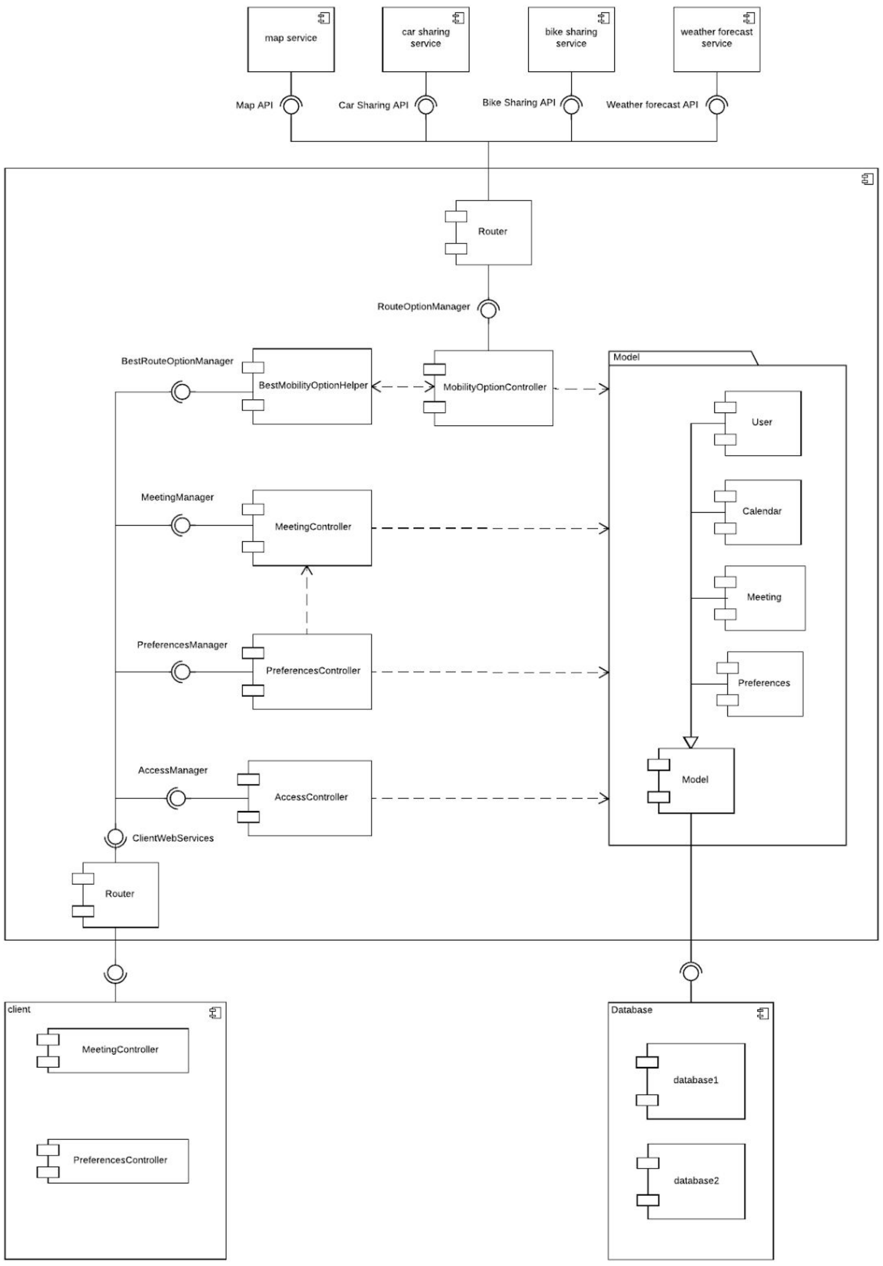


Figure 5

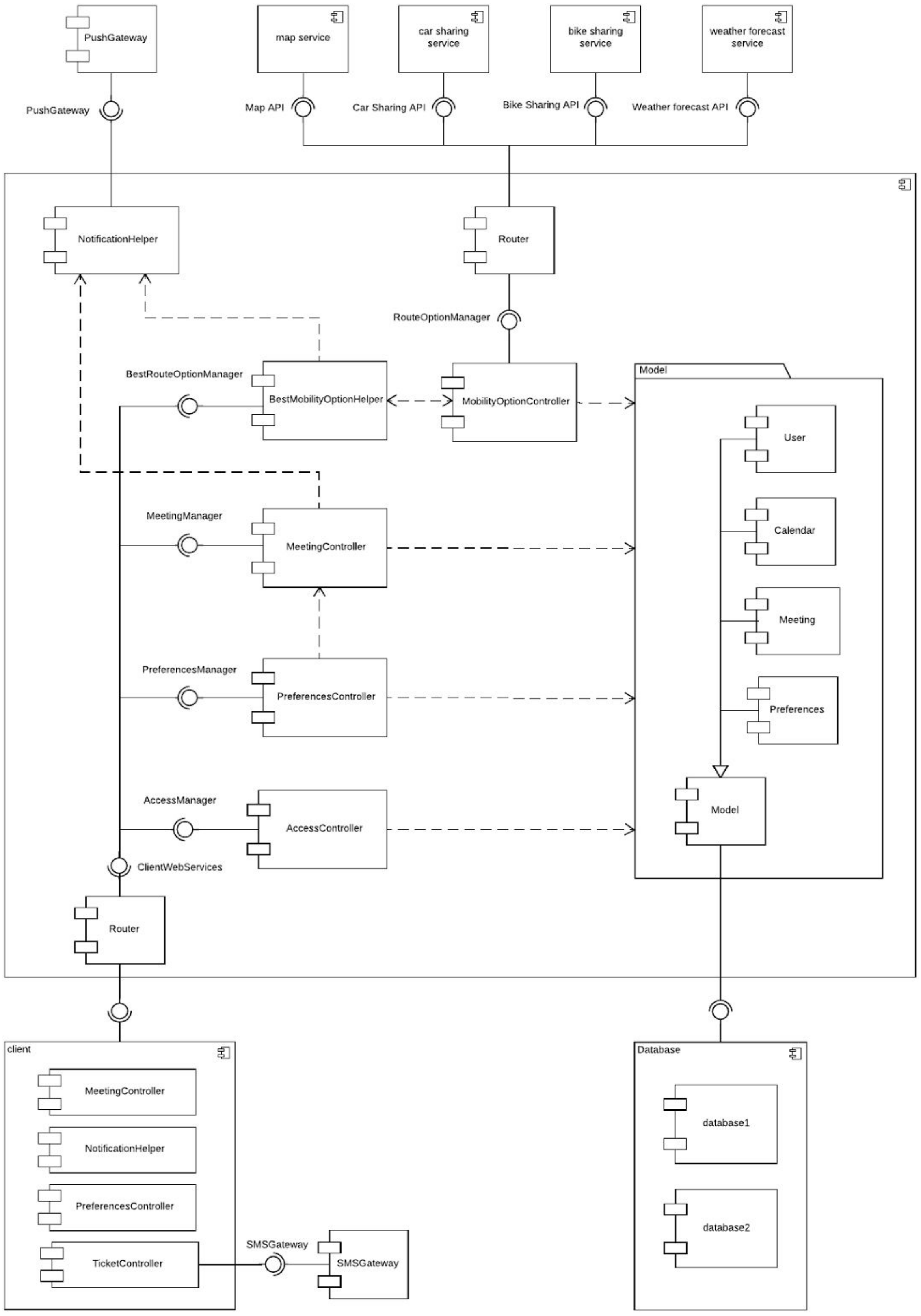

PushGateway
map service
car sharing service
bike sharing service
weather forecast service
PushGateway
Map API
Car Sharing API
Bike Sharing API
Weather forecast API
NotificationHelper
Router
RouteOptionManager
BestRouteOptionManager
BestMobilityOptionHelper
MobilityOptionController
Model
User
Calendar
Meeting
Preferences
Model
MeetingManager
MeetingController
PreferencesManager
PreferencesController
AccessManager
AccessController
ClientWebServices
Router
client
MeetingController
NotificationHelper
PreferencesController
TicketController
SMSGateway
SMSGateway
Database
database1
database2

Figure 6

7. EFFORT SPENT

Day	Probo Giovanni	Tommasi Giovanni
1 novembre	5 h	5 h
3 novembre	2.5 h	2 h
5 novembre	3 h	4 h
6 novembre	3 h	-
8 novembre	2 h	2.5 h
9 novembre	-	2 h
10 novembre	2 h	-
12 novembre	3 h	2.5 h
14 novembre	1h	-
15 novembre	-	3 h
16 novembre	2 h	2.5 h
19 novembre	3.5 h	2.5 h
21 novembre	1.5 h	5 h
23 novembre	3 h	3 h
24 novembre	1.5 h	1.5 h
25 novembre	3 h	3 h

8. REFERENCES

- Specification document : Mandatory Project Assignments.pdf

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

- IEEE Std 1016 tm -2009 Standard for Information Technology-System Design-Software Design Descriptions.