1. INTRODUCTION
A. Purpose
B. Scope
C. Definitions, Acronyms, Abbreviations
D. Revision history
E. Reference Documents
F. Document Structure

2. ARCHITECTURAL DESIGN
A. Overview: High-level components and their interaction SCHEMA FATTO BENE
C. Component view
D. Deployment view
E. Runtime view: You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases
F. Component interfaces
G. Selected architectural styles and patterns: Please explain which styles/patterns you used, why, and how (CLOUD PER ECONOMICITA' ECC)
H. Other design decisions

3. ALGORITHM DESIGN: Focus on the definition of the most relevant algorithmic part

4. USER INTERFACE DESIGN: Provide an overview on how the user interface(s) of your system (bce and ux diagram + rasd mockups)
will look like; if you have included this part in the RASD, you can simply refer to what you have
already done, possibly, providing here some extensions if applicable.

5. REQUIREMENTS TRACEABILITY: Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.

7. IMPLEMENTATION, INTEGRATION AND TEST PLAN: Identify here the order in which you plan
to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.

8. EFFORT SPENT: In this section you will include information about the number of hours each
group member has worked for this document.

9. REFERENCES

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

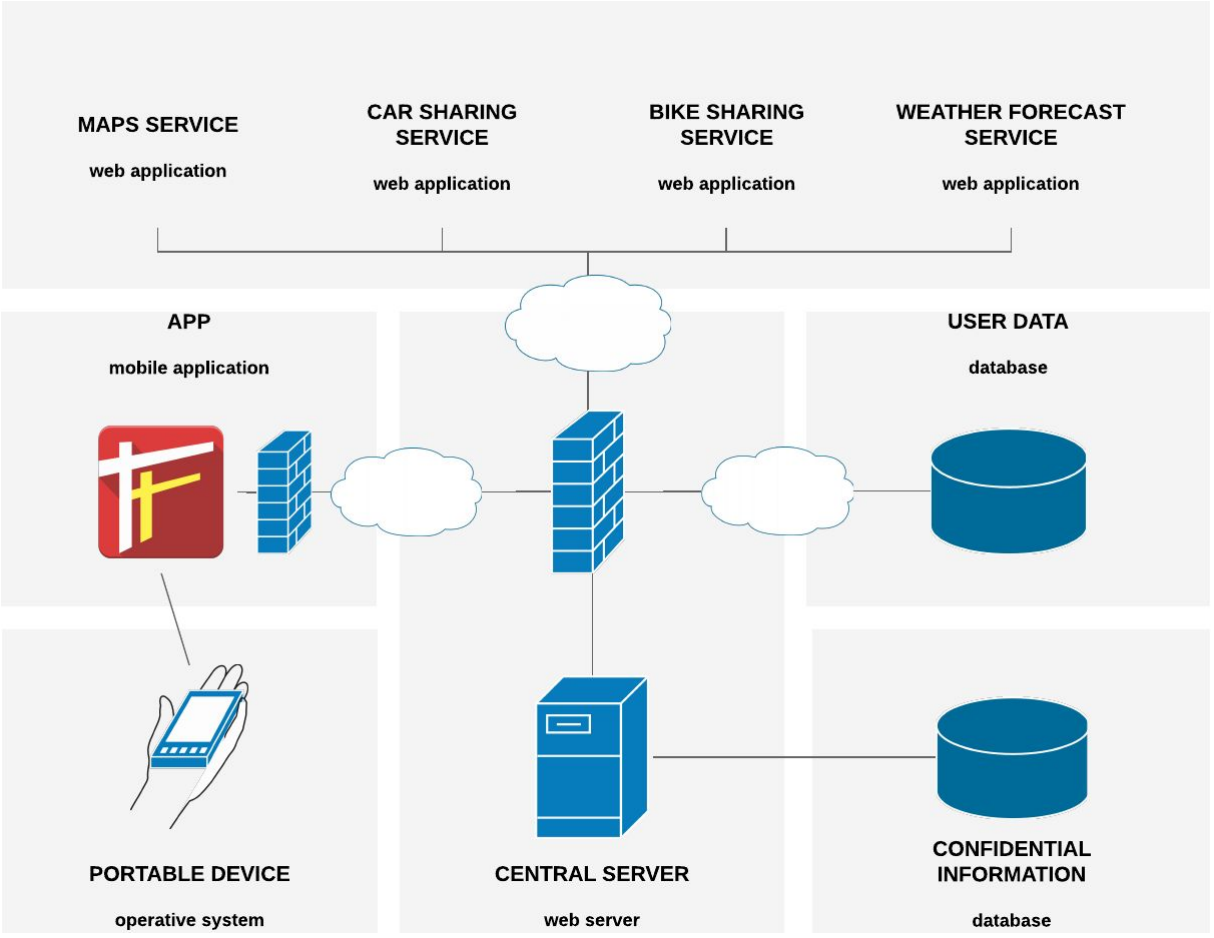
1.E - REFERENCE DOCUMENTS

1.F - DOCUMENT STRUCTURE

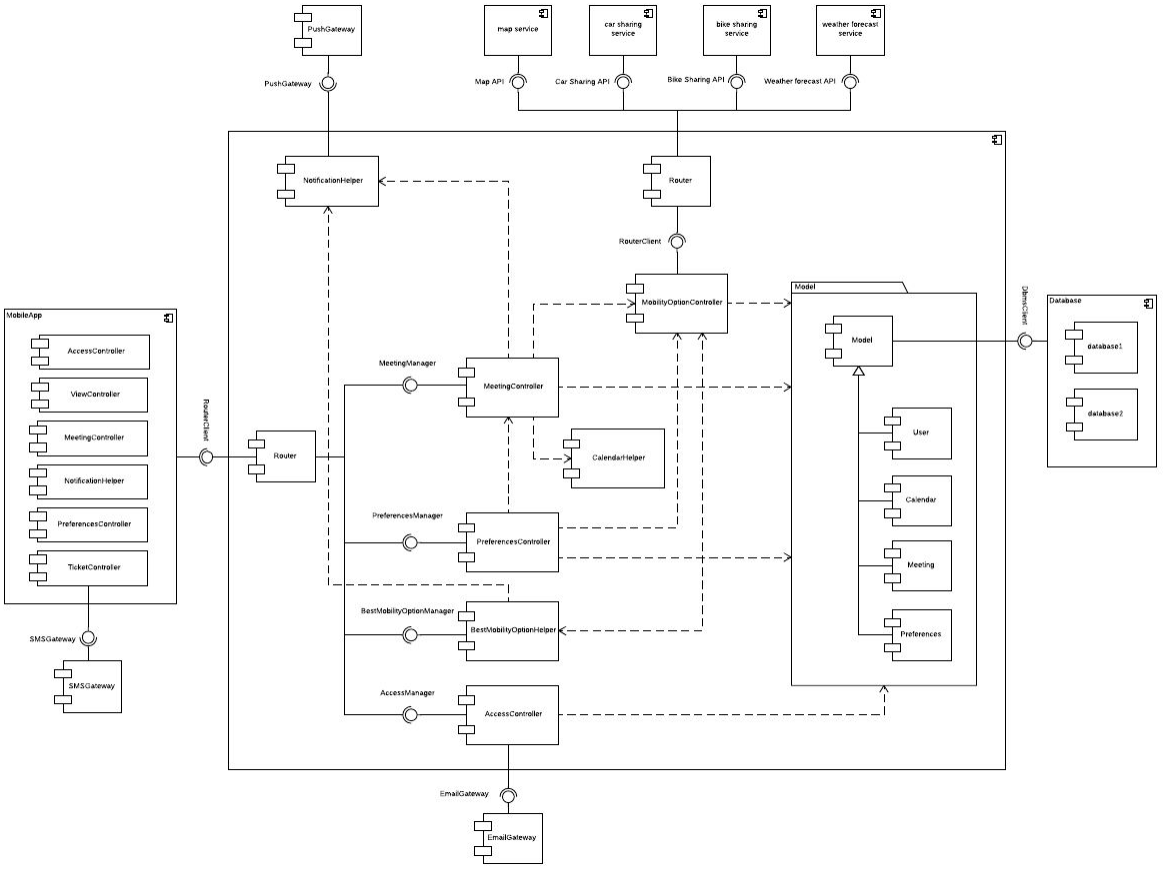
2. ARCHITECTURAL DESIGN

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.

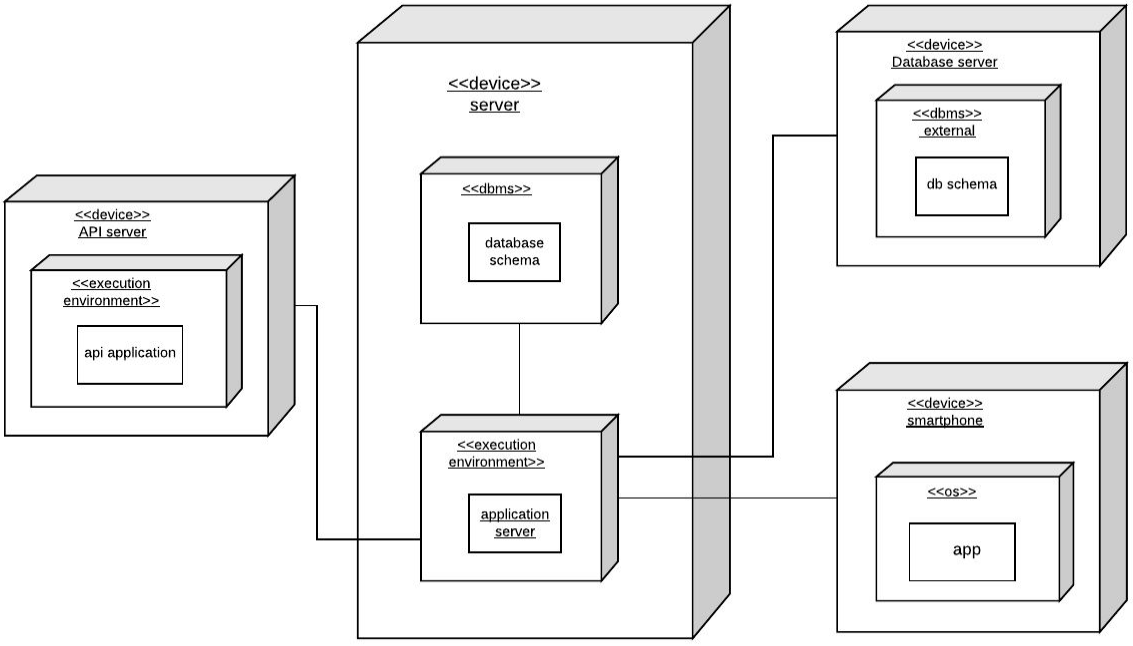


2.B - COMPONENT VIEW



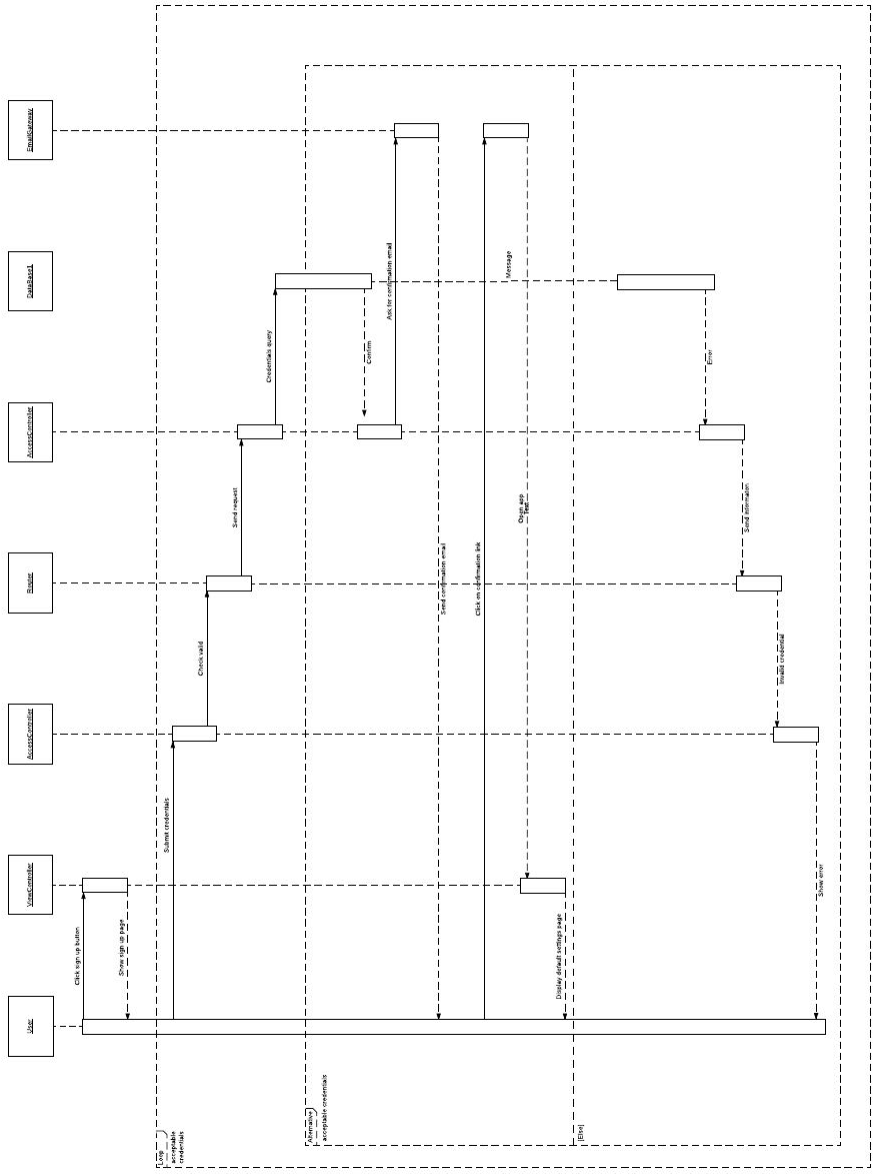
2.C - DEPLOYMENT VIEW

descrizione!

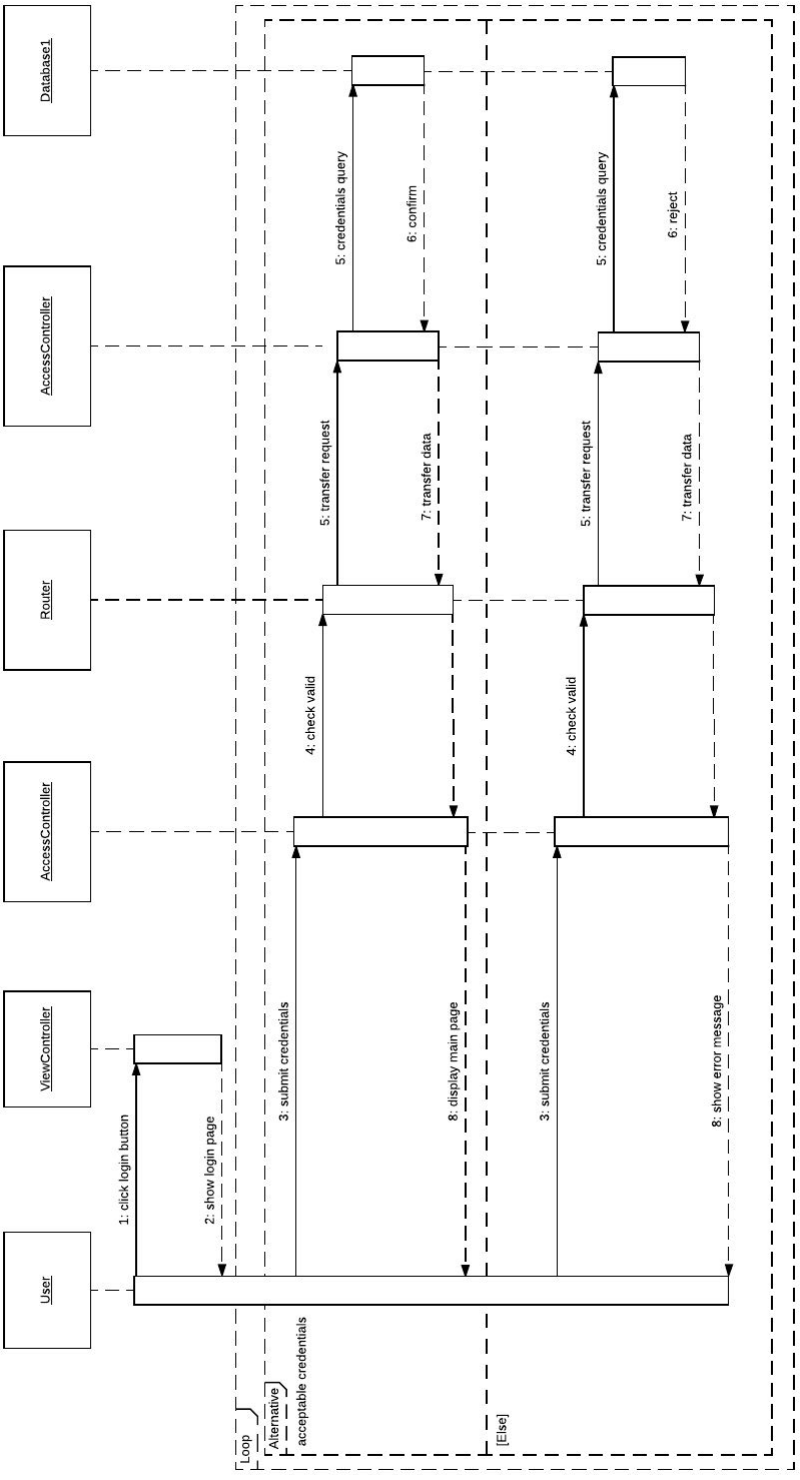


2.D - RUNTIME VIEW

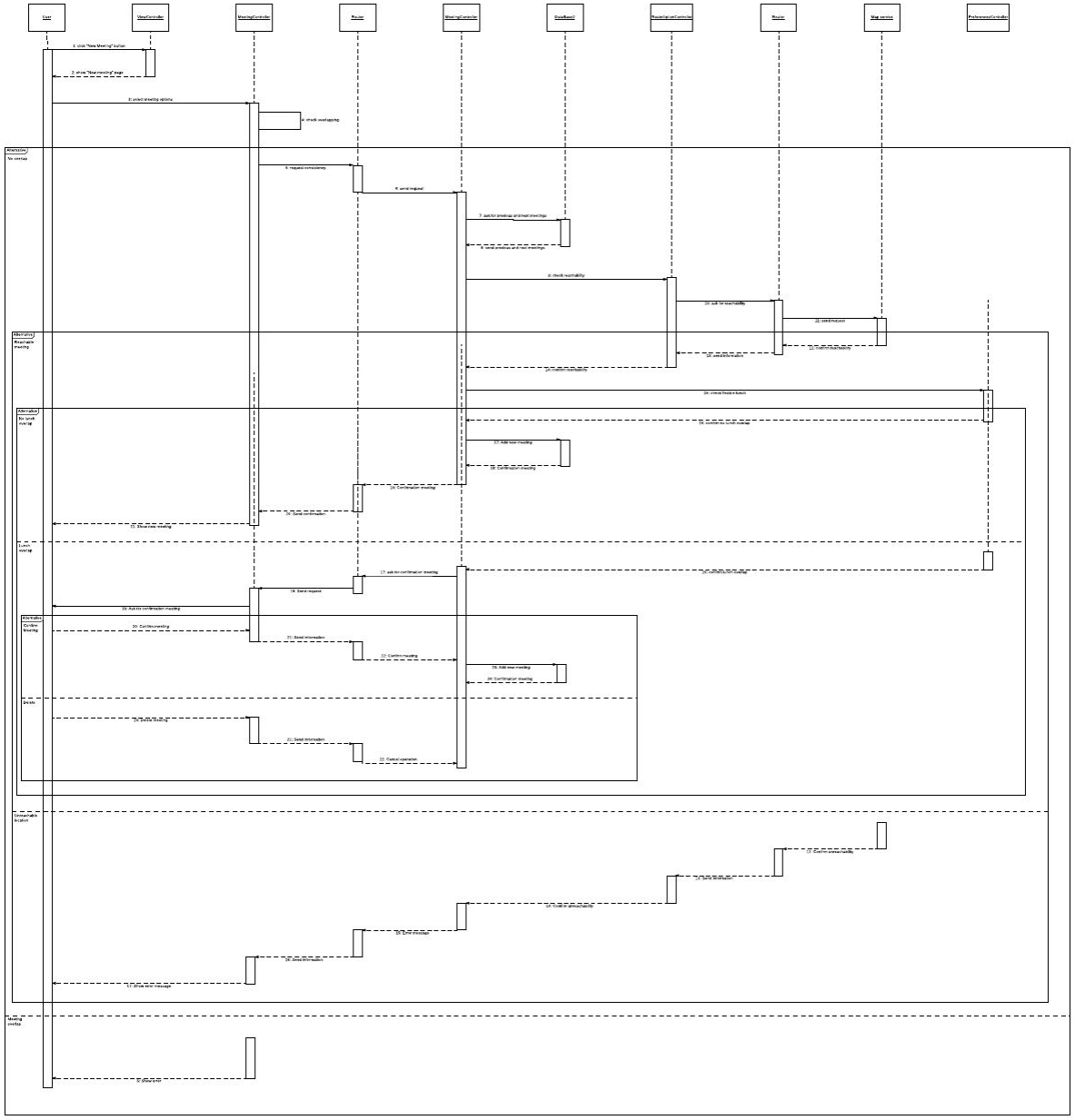
2.D.1 - REGISTRATION PROCESS



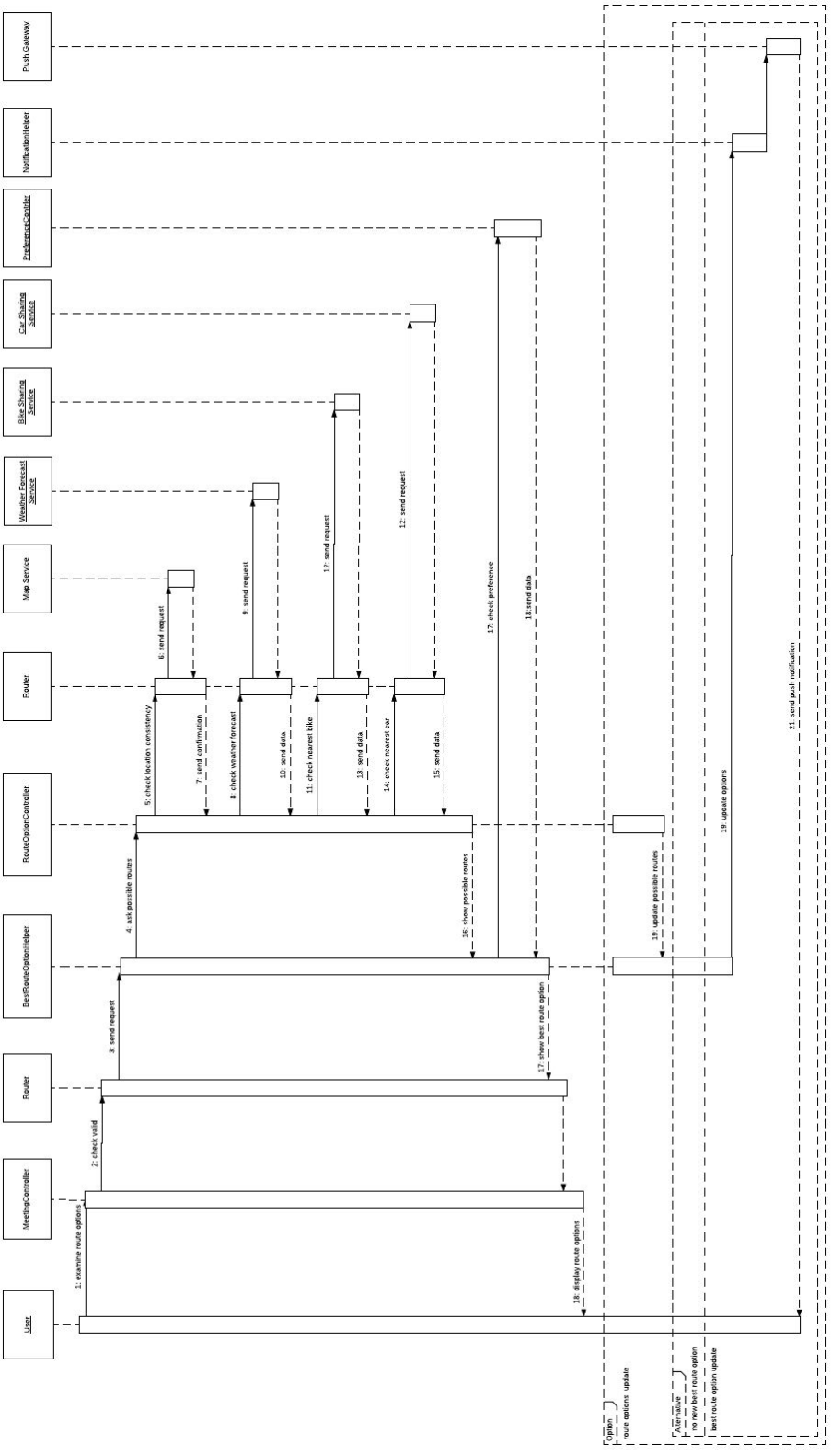
2.D.2 - LOGIN


Database1
AccessController
Router
AccessController
ViewController
User
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
Loop
Alternative
acceptable credentials
[Else]
3: submit credentials
4: check valid
5: transfer request
5: credentials query
6: reject
7: transfer data
8: show error message

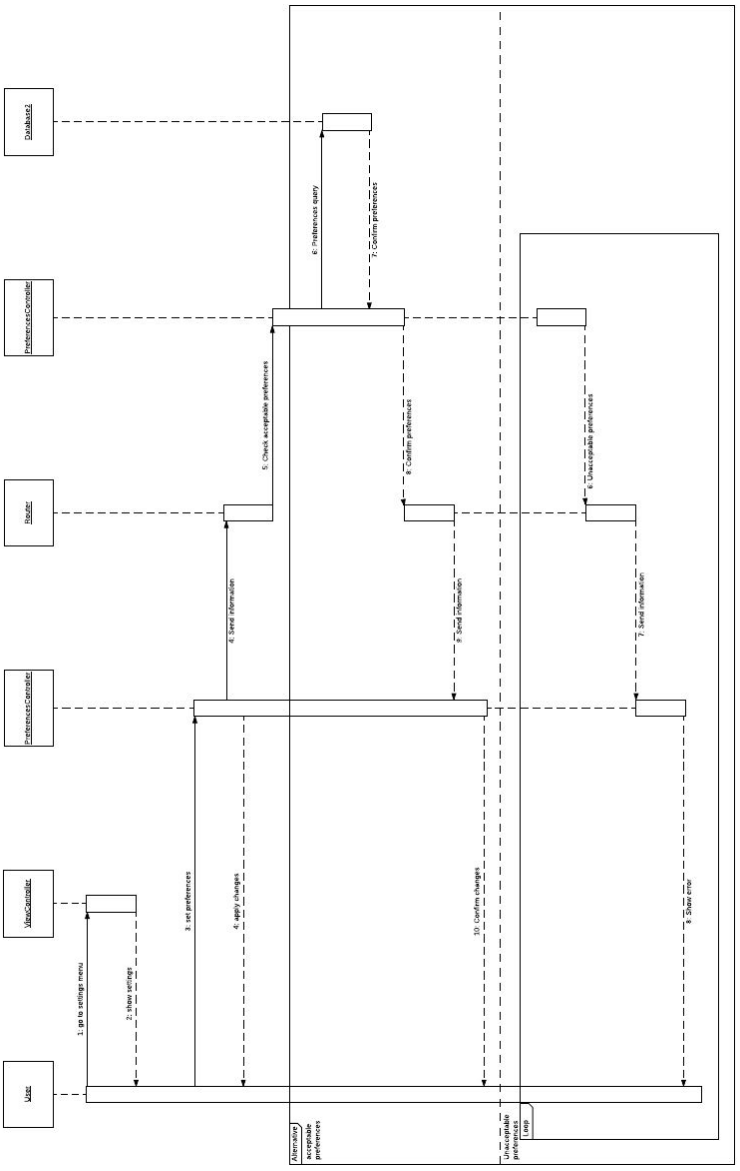
2.D.3 - CREATION OF A MEETING



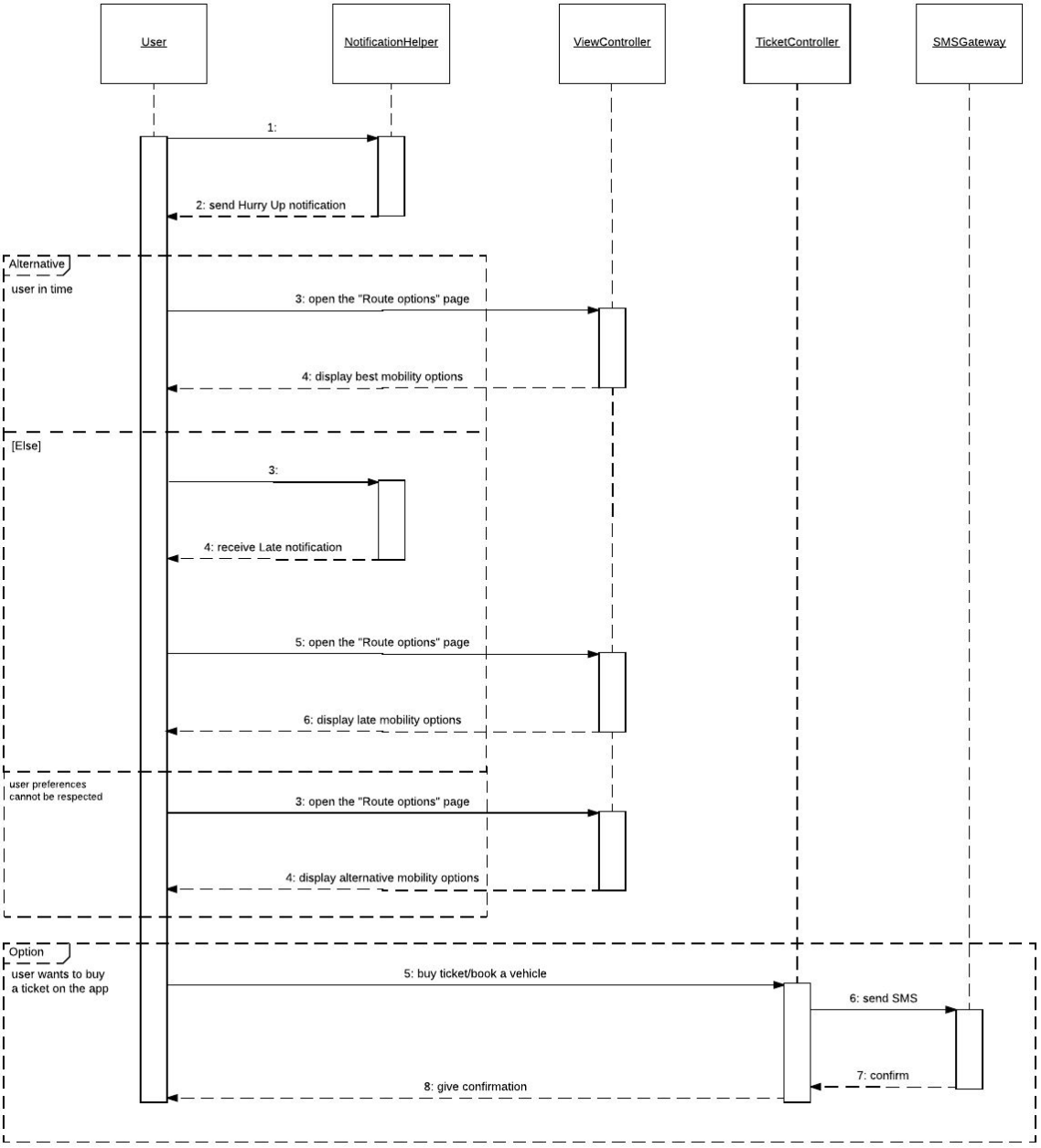
2.D.4 - EXAMINE BEST MOBILITY OPTIONS



2.D.5 - SETTING PREFERENCES

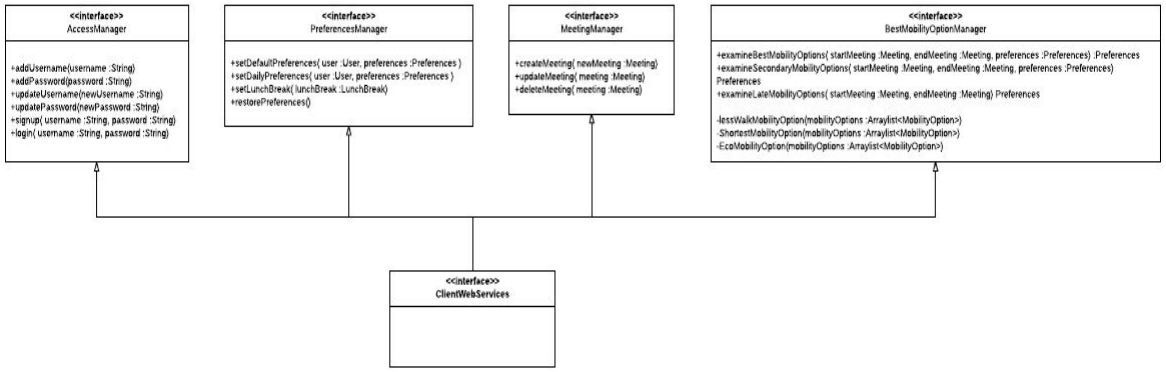


2.D.6 - STARTING A JOURNEY

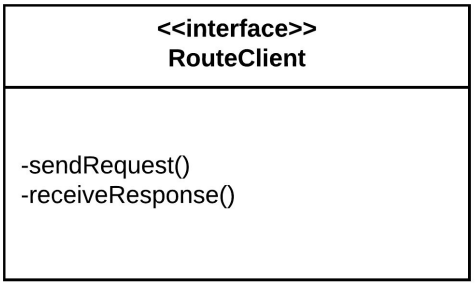


2.F - COMPONENT INTERFACES

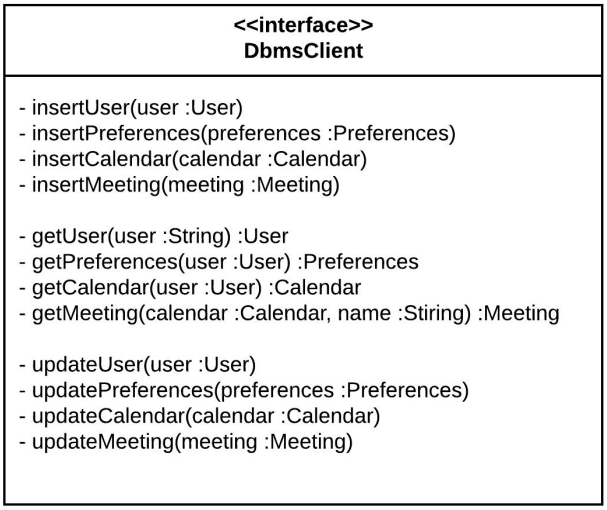
2.F.1 - WEB SERVER INTERFACES



2.F.2 - ROUTER INTERFACE



2.F.3 - DATABASE INTERFACE



3. ALGORITHM DESIGN

```
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;
}
```

```
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;
}
```

```
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;
};
```