

POLITECNICO DI MILANO



Corso di Laurea Magistrale in Computer Science and Engineering  
Dipartimento di Elettronica e Informazione

---

# Travlendar+

## *Design Document*

---

Reference professor:  
Prof. Elisabetta Di Nitto

Authors:  
Alessandro Aimi Matr. 899642  
Roberto Bigazzi Matr. 899411  
Filippo Collini Matr. 829918

*Version 1.0.0*  
*26/11/2017*  
Academic Year 2017-2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.A	Description of the given problem . . . . .	1
1.B	Definitions and Acronyms . . . . .	2
1.B.1	Definitions . . . . .	2
1.B.2	Acronyms . . . . .	2
1.C	Revision History . . . . .	2
1.D	References . . . . .	2
1.E	Document Structure . . . . .	3
1.F	Used tools . . . . .	3
<b>2</b>	<b>Architectural Design</b>	<b>4</b>
2.A	Overview . . . . .	4
2.B	Component View . . . . .	5
2.C	Deployment View . . . . .	6
2.D	Runtime View . . . . .	6
2.E	Component Interfaces . . . . .	11
2.F	Selected Architectural Styles and Patterns . . . . .	11
2.G	Other Design Decisions . . . . .	11
<b>3</b>	<b>Algorithm Design</b>	<b>12</b>
<b>4</b>	<b>User Interface Design</b>	<b>25</b>
4.A	Mockups . . . . .	25
4.B	UX Diagrams . . . . .	34
4.C	BCE Diagrams . . . . .	37
<b>5</b>	<b>Requirements Traceability</b>	<b>38</b>
<b>6</b>	<b>Implementation, Integration and Test Plan</b>	<b>39</b>
6.A	Implementation Order . . . . .	39

---

6.B	Integration and Test Plan . . . . .	40
6.B.1	Entry criteria . . . . .	40
6.B.2	Elements to Be Integrated . . . . .	40
6.B.3	Integration Strategy . . . . .	40
<b>7</b>	<b>Effort Spent</b>	<b>41</b>

# Chapter 1

## Introduction

This document is the Design Document (DD) of a mobile application called Travlendar+. It is mainly addressed to the software development team and its purpose is to provide an overall view of the architecture of the new system. The document defines:

- High level architecture;
- Used design patterns;
- Main components of the system;
- Runtime behavior.

### 1.A Description of the given problem

Travlendar+ is a mobile, calendar-based application that helps the user to manage his appointments and to a greater extent set up the trip to his destination, choosing the best means of transport depending on his needs. Travlendar+ will choose the most suitable way to get the user to his destination between a large pool of options, considering public transportation, personal vehicles, locating cars or bikes of sharing services and walking to the destination. It will take account of weather, traffic, possible passengers if any, the user-set break times and the potential will to minimize the carbon footprint of the trip, always focusing on taking him on time to his scheduled appointments.

Eventually the user will be able to purchase the tickets he will use to reach his destination in-app. The great customizability is one of the main strengths of Travlendar+, being able to fully comply with the user needs.

## 1.B Definitions and Acronyms

### 1.B.1 Definitions

List of the definitions used in this paper:

- \*\*\*\*\*

### 1.B.2 Acronyms

List of the acronyms used in this paper:

- RASD: Requirements analysis and specification document;
- DD: Design document;
- MOT: Means of transport;
- RMI: Remote Method Invocation;
- API: Application Programming Interface;
- UX: User experience;
- BCE: Boundary Control Entity;
- UI: User interface.

## 1.C Revision History

- 26/11/2017 Version 1.0.0 - First complete drawing up of the DD;

## 1.D References

Documents list:

- Mandatory Project Assignments.pdf

Online sites list:

- <https://developers.google.com/maps/> (used also to design mockups)
- <https://developers.google.com/google-apps/calendar/> (used also to design mockups)
- <https://openweathermap.org/api>

## 1.E Document Structure

The paper is structured as follows:

- Chapter 1: Introduction to the document, including some information about its composition and the description of the problem;
- Chapter 2: This chapter shows the main components of the system and the relationships between them. It also explains the main architectural styles and patterns adopted in the design of the system;
- Chapter 3: This chapter explains how the system will work using algorithms. Java code is used to write down the most significant algorithms for the application;
- Chapter 4: This chapter shows mockups of the application and more details about the User Interface using UX and BCE diagrams;
- Chapter 5: This chapter explains how the decisions taken in the RASD are associated to design decisions;
- Chapter 6: This chapter describes the order of implementation of the subcomponents of the system and the order in which integrate such subcomponents and test the integration;
- Chapter 7: Effort spent by the authors to draw up the document.

## 1.F Used tools

The tools used to create this document are:

- Photoshop for mockups;
- Draw.io for diagrams;
- Atom for algorithms drawing up;
- Github as version controller and to share documents;
- LaTeX for typesetting this document;
- Texmaker as editor;

# Chapter 2

## Architectural Design

### 2.A Overview

Architectural design is of crucial importance in software engineering because it will have to take account of functional and non-functional requirements, to meet the stakeholders needs and requests, and to help not to focus only on standalone elements losing the so called big picture of the system, always adhering to general principles of good quality. An important aspect is in fact to find a good trade-off between the high-level description near to the analysis and the low-level one near to the implementation. Coming up with good quality design and architecture is mostly a matter of experience and in our field, is also known the importance of the reusability of other's people work. So, we tried to build our system with various kind of this patterns and known architectural styles.

## 2.B Component View

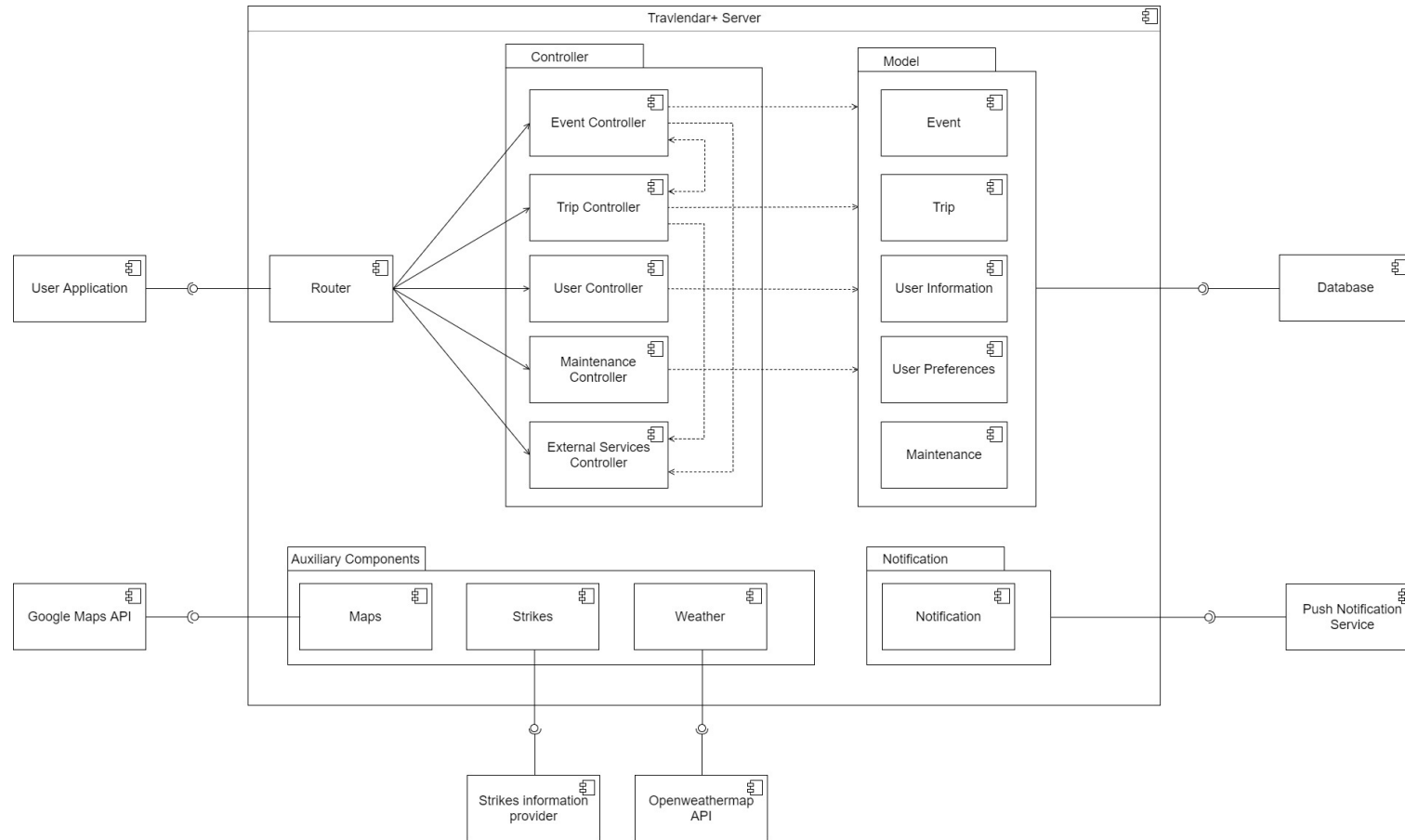


Figure 2.1: Component View of the system



## 2.C Deployment View

## 2.D Runtime View

In this section some sequence diagrams will be presented to describe the interactions that happen between the main components of the system when the most common functionalities are used. This is a high-level description of the actual interactions of the system-to-be, so functions and their names may be added, modified or deleted during the development process.

The functionalities considered for the runtime view are: login, adding of an event on the calendar, trip planning and trip customization.

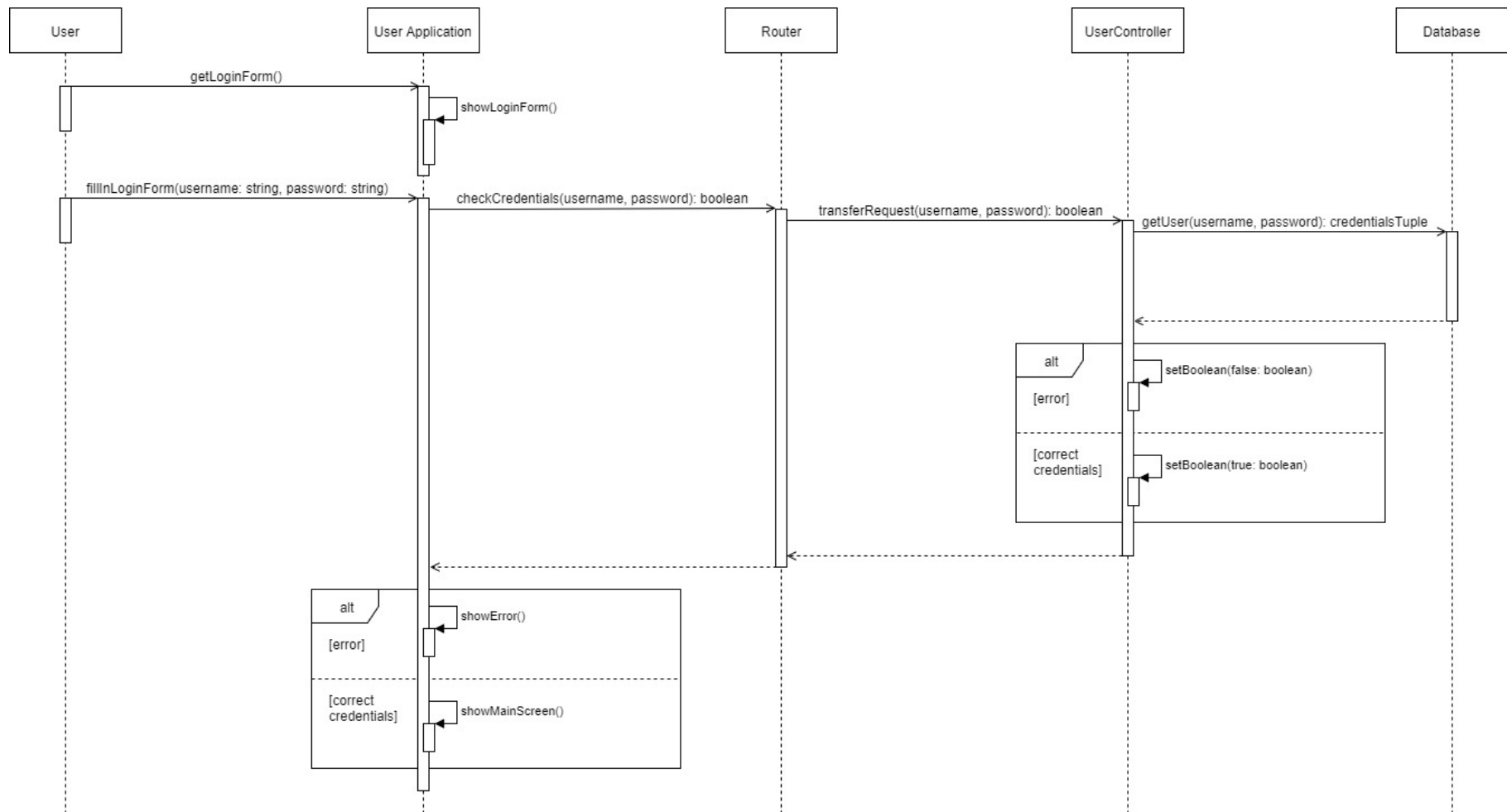


Figure 2.2: User Login runtime view

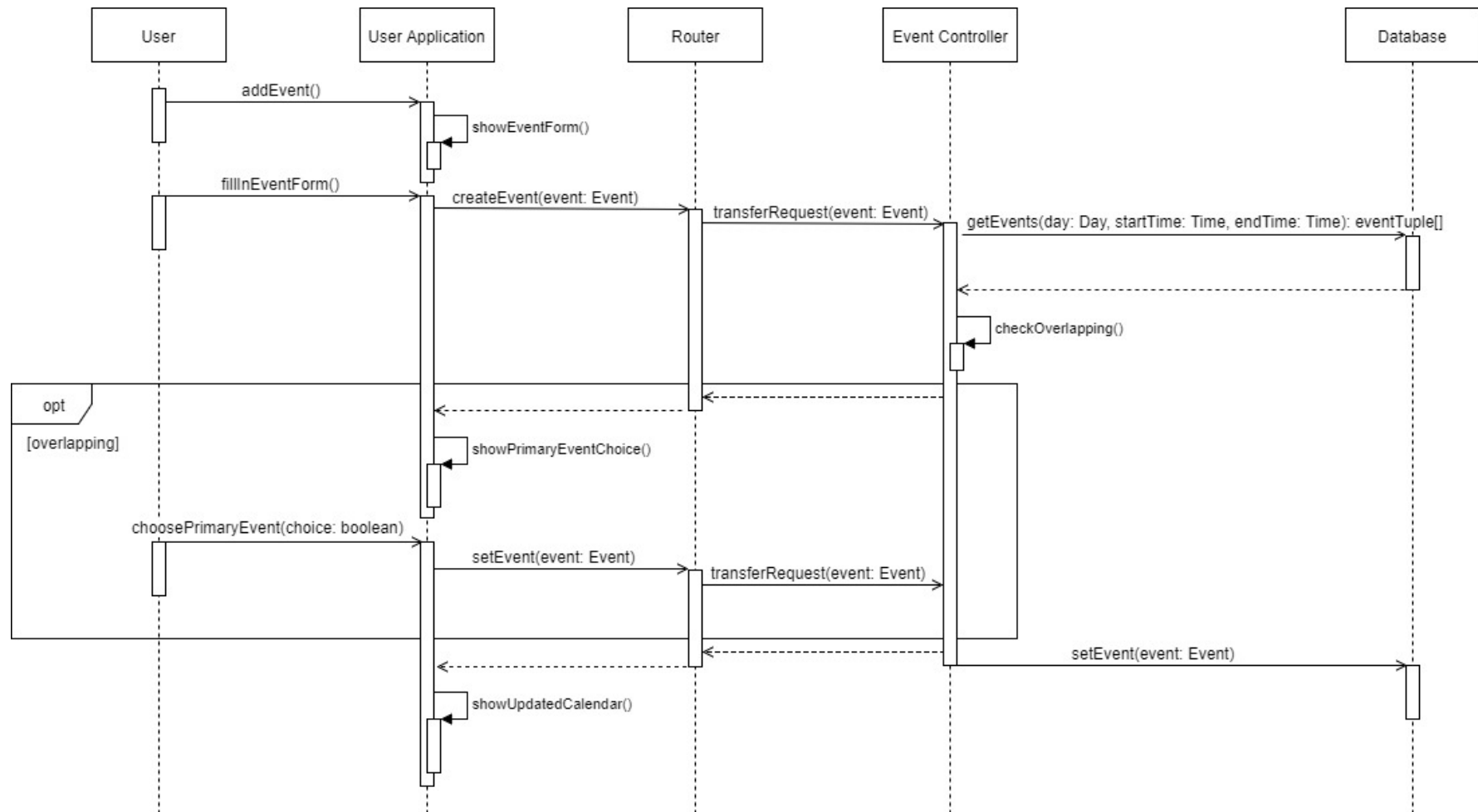


Figure 2.3: Adding of an event in the calendar runtime view

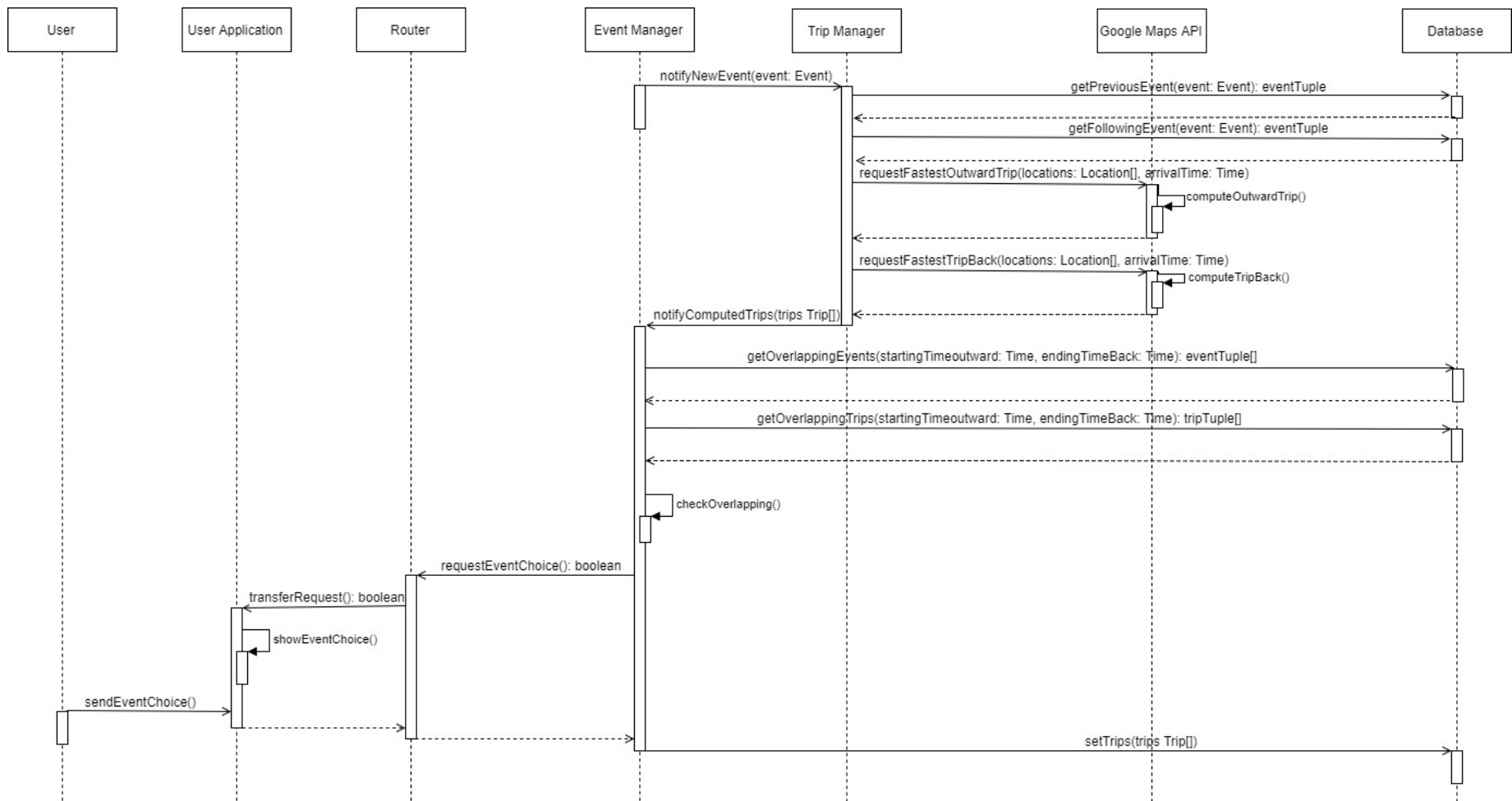


Figure 2.4: Trip planning runtime view

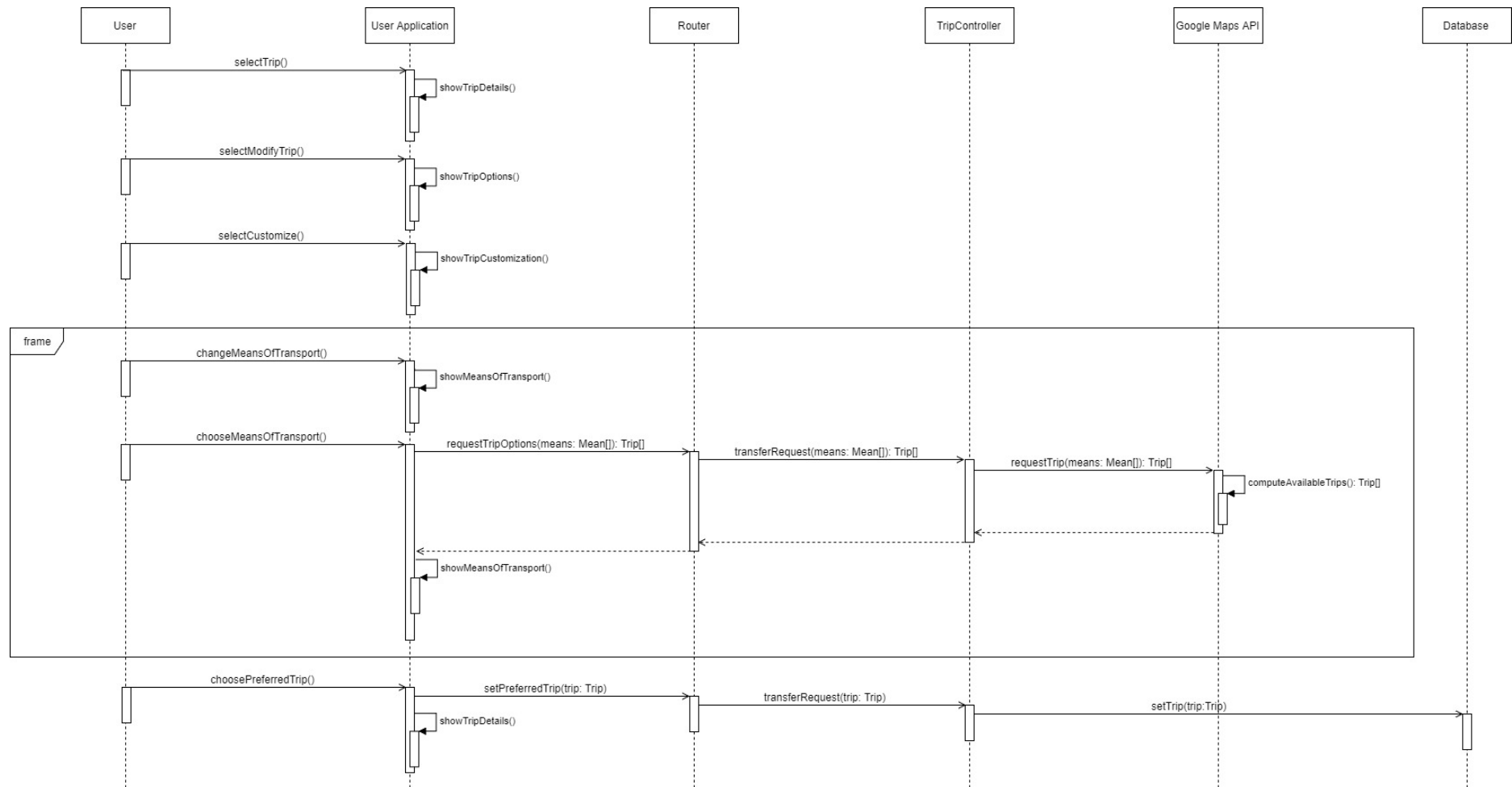


Figure 2.5: Trip customization runtime view

**2.E Component Interfaces**

**2.F Selected Architectural Styles and Patterns**

**2.G Other Design Decisions**

# Chapter 3

## Algorithm Design

In this section we show the idea behind the algorithms structure of the core functions of the system. We choose to elaborate on the processes concerning the creation of a new event, the update of the planned trips of all user's events, the trip planning, the calculation of the travel time for shared means of transport (due to the fact that it is not directly obtainable from Google Maps API) and the users's dynamic events scheduling, which are the main objectives of the system. After the user inputs the data of a new event, the system will first run the overlapping check algorithm, which will then run the trips update algorithm to keep them up to date. This second process is of great importance, and will be run every time there is a change in preferences, events or trips, in addition of a daily update. The main block of the update algorithm is the trip planning one, checking the feasibility of the trips and suggesting the best way to reach an event. This one will be run more frequently than usual when it is almost time to start a trip. We decided to show a flow chart with processes, serving as code description to ease the understanding, due to the more schematic form in blocks. The code is written in a pseudo java inspired on the probable class structure of the system. The objective is to show the reader the logic under the decision precesses of our sistem, made to be as practical and human-like as possible when proposing the beat travel solution.

## Overlapping check algorithm

The event creator  
initialize two blank  
trips before (ending  
on the start of the event)  
and after (beginning  
on the end of the event)

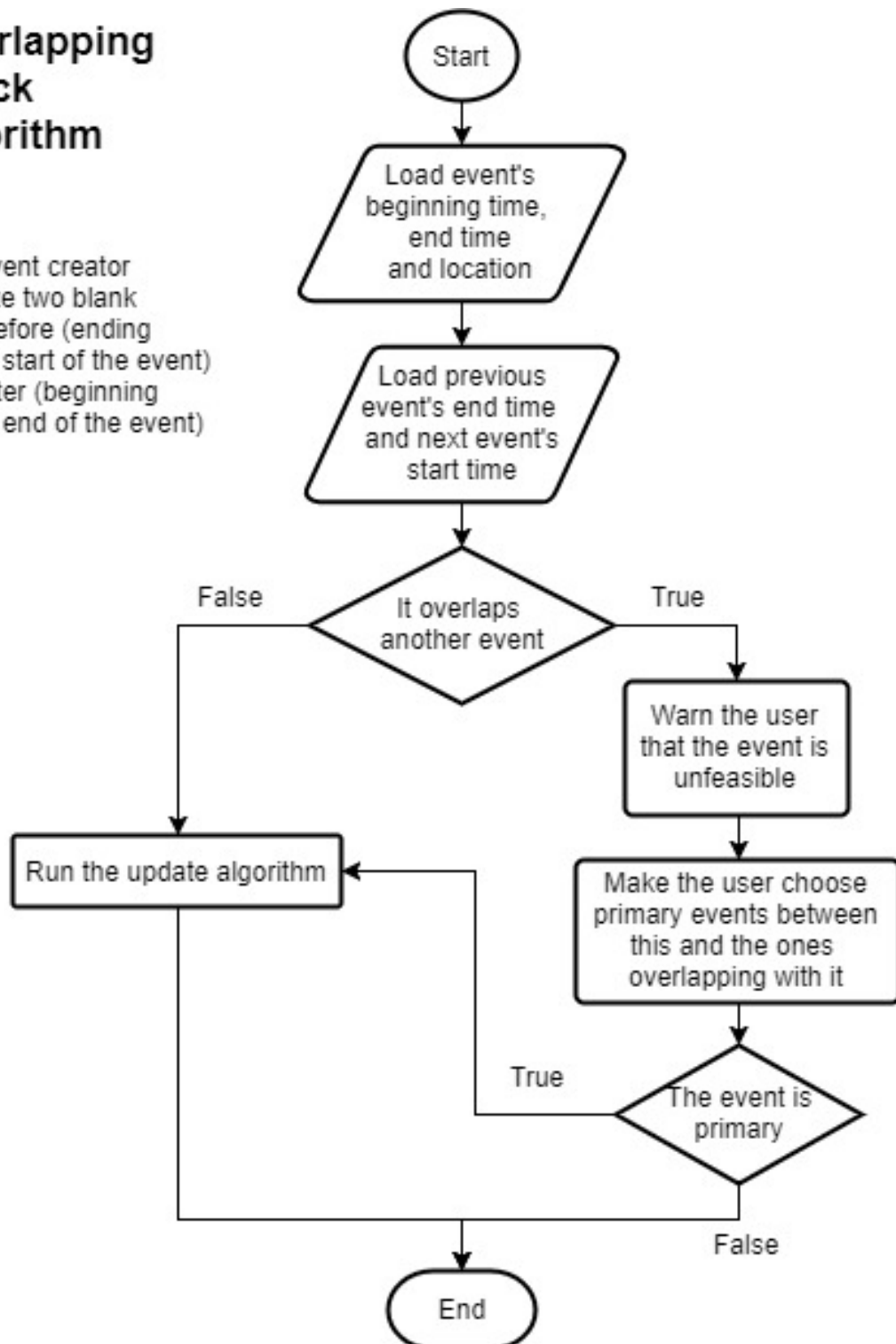


Figure 3.1: Overlap check algorithm



```

1
2  /*
3  *   Event is already initialized with to/from trip;
4  *   by default the trip to the event starts from home
5  *   and it ends on event location , instead the trip from
6  *   the event starts from event location and goes to home.
7  */
8
9  public void overlappingCheckAlgorithm(Event event){
10     ArrayList<Event> overlap = personalEventsController .
        getOverlappingEvents(event); // in SQL !(( TStart1>=TEnd)
        ||(TEnd1<=TStart))
11     if (!overlap.isEmpty()){
12         overlappingWarning(event);
13         if(getPrimaryEventChoice(event,overlap)){ // 1 if the event
            is chosen as primary instead of the ones it overlaps
            with
14             personalEventsController.removePrimaryEvent(overlap);
15             personalEventsController.addSecondaryEvent(overlap);
16             personalEventsController.addPrimaryEvent(event);
17             tripsUpdateAlgorithm();
18         } else {
19             personalEventsController.addSecondaryEvent(event);
20         }
21     } else {
22         personalEventsController.addPrimaryEvent(event);
23         tripsUpdateAlgorithm();
24     }
25 }

```

## Trip update algorithm

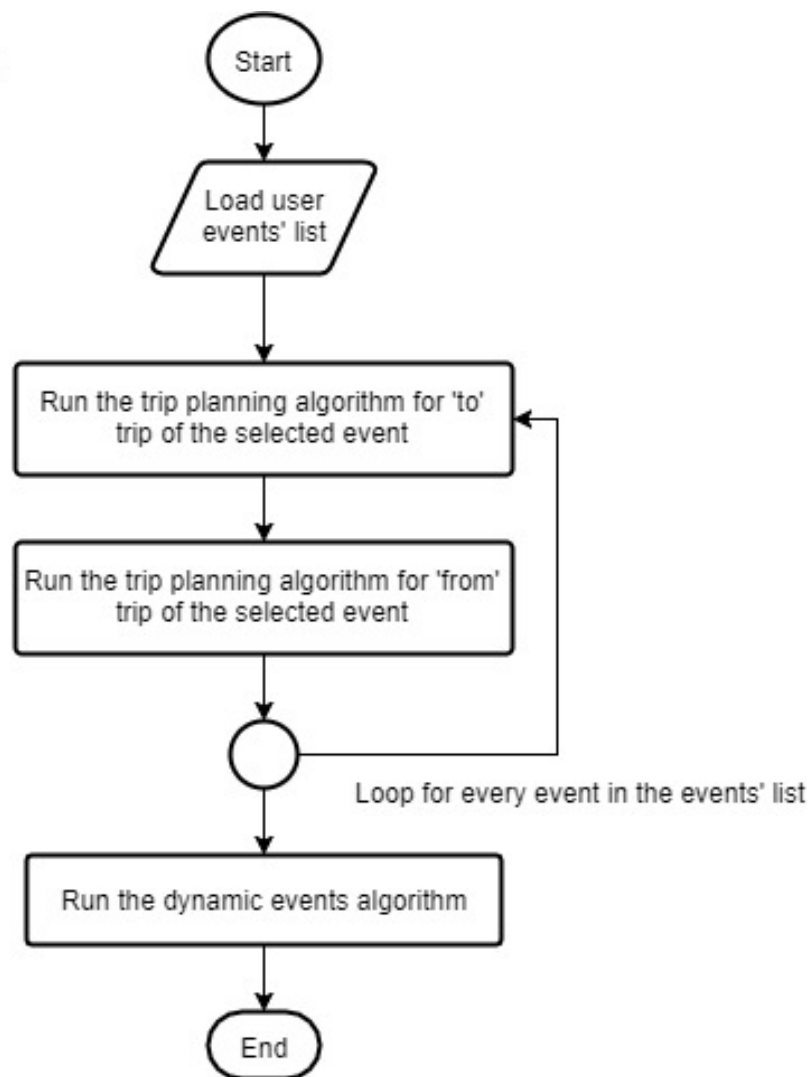


Figure 3.2: Trip update algorithm

```
1 public void tripsUpdateAlgorithm() {  
2     for(Event e : personalEventsController.getEventsList()) {  
3         tripPlanningAlgorithm(e.getToTrip(), e);  
4         tripPlanningAlgorithm(e.getFromTrip(), e);  
5     }  
6     dynamicScheduleAlgorithm();  
7 }
```

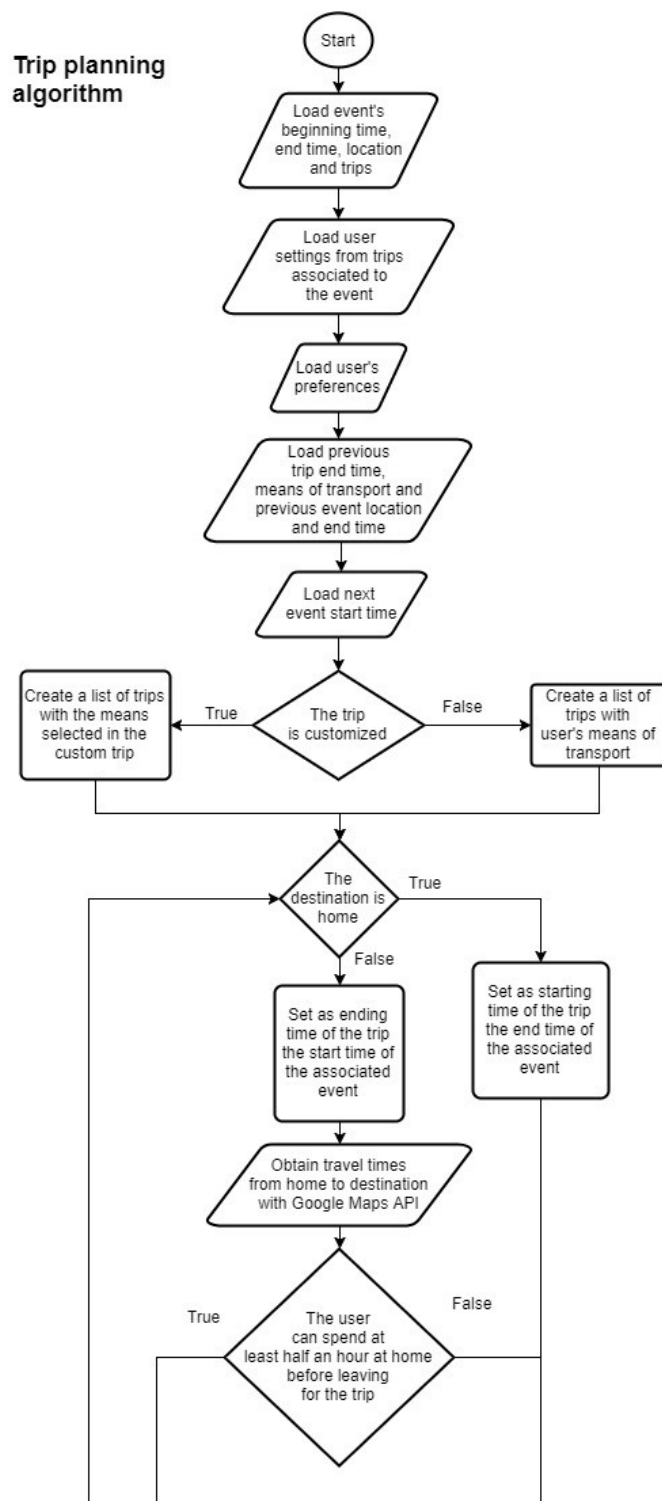


Figure 3.3: Trip planning algorithm part 1

### 3. ALGORITHM DESIGN

---

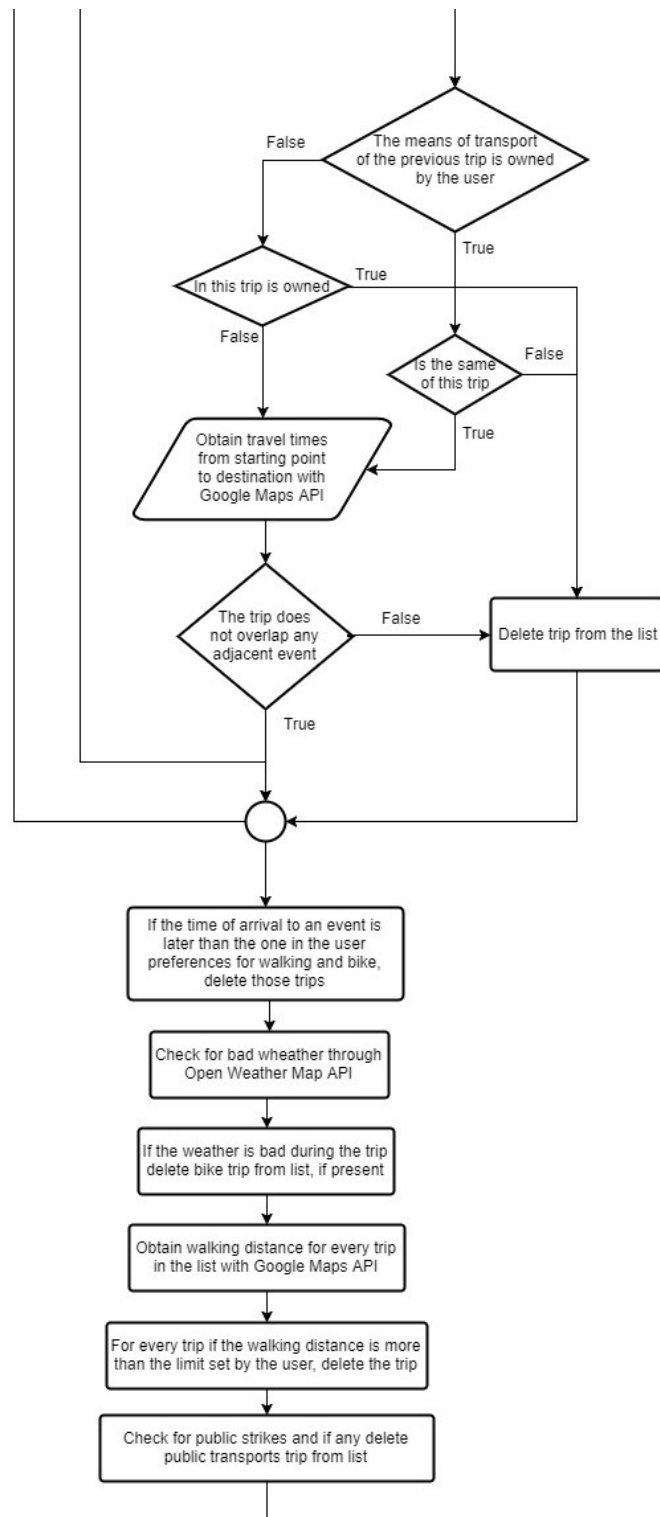


Figure 3.4: Trip planning algorithm part 2

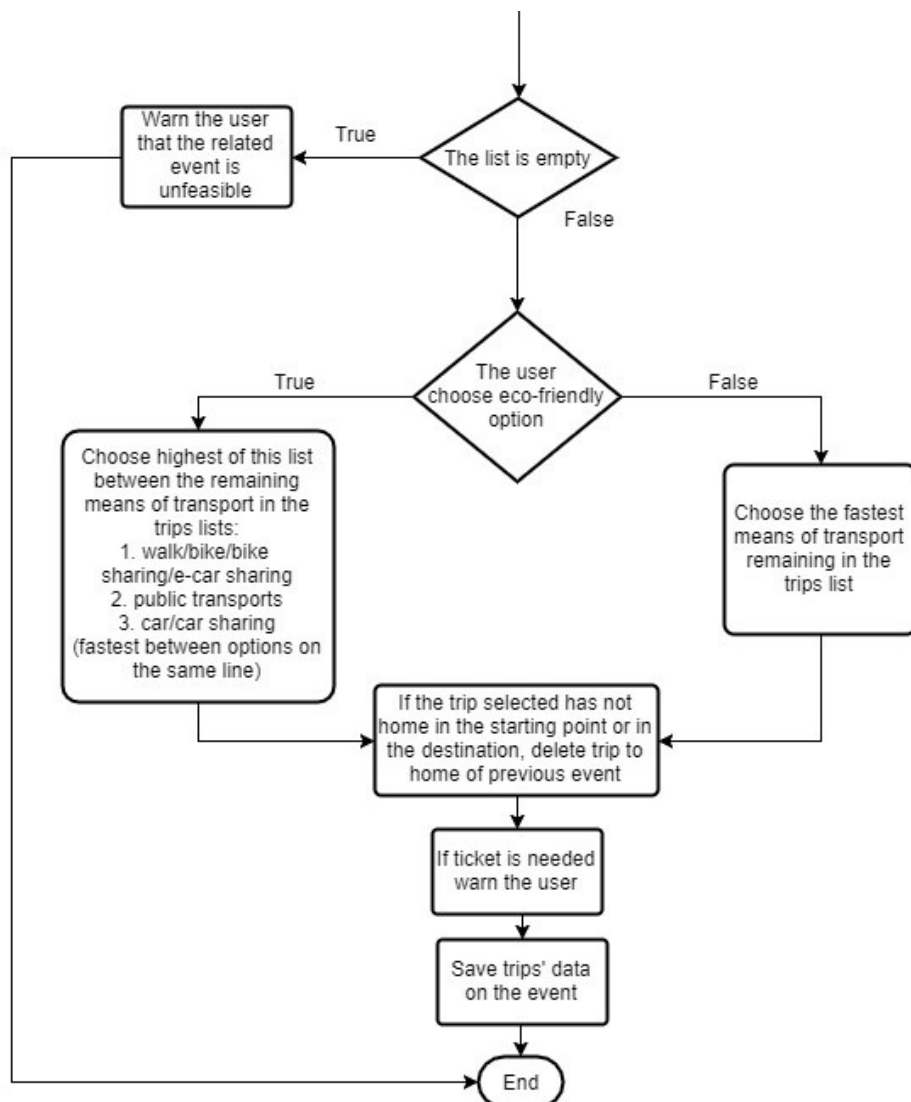


Figure 3.5: Trip planning algorithm part 3

### 3. ALGORITHM DESIGN

---

```
1
2 public void tripPlanningAlgorithm(Trip t, Event e){
3
4     ArrayList<Trip> trips = new ArrayList<>();
5     ArrayList<MeansOfTransport> motList;
6     Trip previousTrip;
7
8     if(t.isCustom()){
9         motList = t.getCustomMOTs();
10    } else {
11        motList = user.getPreferences().getMOTList();
12    }
13
14    for(MeansOfTransport mot : motList){
15        Trip t1 = new Trip(t,mot);
16
17        if(user.getHomeLocation().isEqual(t.getDestination())){
18            previousTrip = e.getToTrip();
19            t1.setStartTime(Time.addMinutes(e.getEndTime(), 5));
20            mapsController.obtainArrivalTimeFromDepartureTime(t1);
21        } else {
22            previousTrip = personalEventsController.getPrevious(e).
23                getFromTrip();
24            t1.setEndTime(Time.addMinutes(e.getStartTime(), -5));
25            mapsController.obtainDepartureTimeFromArrivalTime(t1);
26
27            if(Time.getBetweenMinutes(t1.getStartTime(), previousTrip.
28                getEndTime()) >= 30){
29                trips.add(t1);
30                break;
31            }
32            t1.setStartingLocation(personalEventsController.
33                getPrevious(e).getLocation());
34            t1.setStartTime(Time.addMinutes(personalEventsController.
35                getPrevious(e).getEndTime(), 5));
36            mapsController.obtainArrivalTimeFromDepartureTime(t1);
37            if(t1.getEndTime().isGreater(Time.addMinutes(e.
38                getStartTime(), -5))){ break; }
39        }
40
41        if(previousTrip.getMOT().isPersonal()){
42            if(!t1.getMOT().isEqual(previousTrip.getMOT())){ break; }
43        } else {
44            if(t1.getMOT().isPersonal()){ break; }
45        }
46
47        if(e.getStartTime().isLaterThan(user.getPreferences.
48            getMaxTimeBikeWalk()) && (t1.getMOT().isEqual(
```

### 3. ALGORITHM DESIGN

---

```
MeansOfTransport.BIKE) || t1.getMOT().isEqual(
MeansOfTransport.WALK))){ break; }
43
44 if(badWeather(t1.getStartingLocation(), t1.getStartTime())
    && t1.getMOT().isEqual(MeanOfTransport.BIKE)){ break; }
45
46 if(mapsController.getWalkDistance(t1) >= user.getPreferences
    ().getMaxWalkDistance()){ break; }
47
48 if(checkStrikes(t1.getStartingLocation(), t1.getStartTime())
    && t1.getMOT().isEqual(MeanOfTransport.PUBLICTRANSPORT)){
    break; }
49
50 trips.add(t1);
51 }
52
53 if(trips.isEmpty()){
54     unfeasibilityWarning(e); // warn the user he cannot come to
        this event in time from the previous
55     personalEventsController.removePrimaryEvent(e);
56     personalEventsController.addSecondaryEvent(e);
57 }
58
59 if(t.isEcoFriendly()){ // t is initialized with the lowest
    EcoPoints (scale assigned to every means of transport) and
    with a travel time of 1 day
60     for(Trip t1 : trips){
61         if(t1.getMOT().getEcoPoint() > t.getMOT().getEcoPoint()){
62             t = t1;
63         } else if(t1.getMOT().getEcoPoint() == t.getMOT().
            getEcoPoint() && t1.getTravelTime() < t.getTravelTime())
            {
64             t = t1;
65         }
66     }
67 } else {
68     for(Trip t1 : trips){
69         if(t1.getTravelTime() > t.getTravelTime()){ t=t1; }
70     }
71
72     if(!(user.getHomeLocation().isEqual(t.getDestination()) ||
        user.getHomeLocation().isEqual(t.getStartingLocation()))){
73         personalEventsController.getPrevious(e).setFromTrip(t);
74     }
75
76     if(t.getMOT().needsTicket()){
77         ticketWarning(t);
78     }
79 }
```

### Sharing means of transport's travel time calculation algorithm

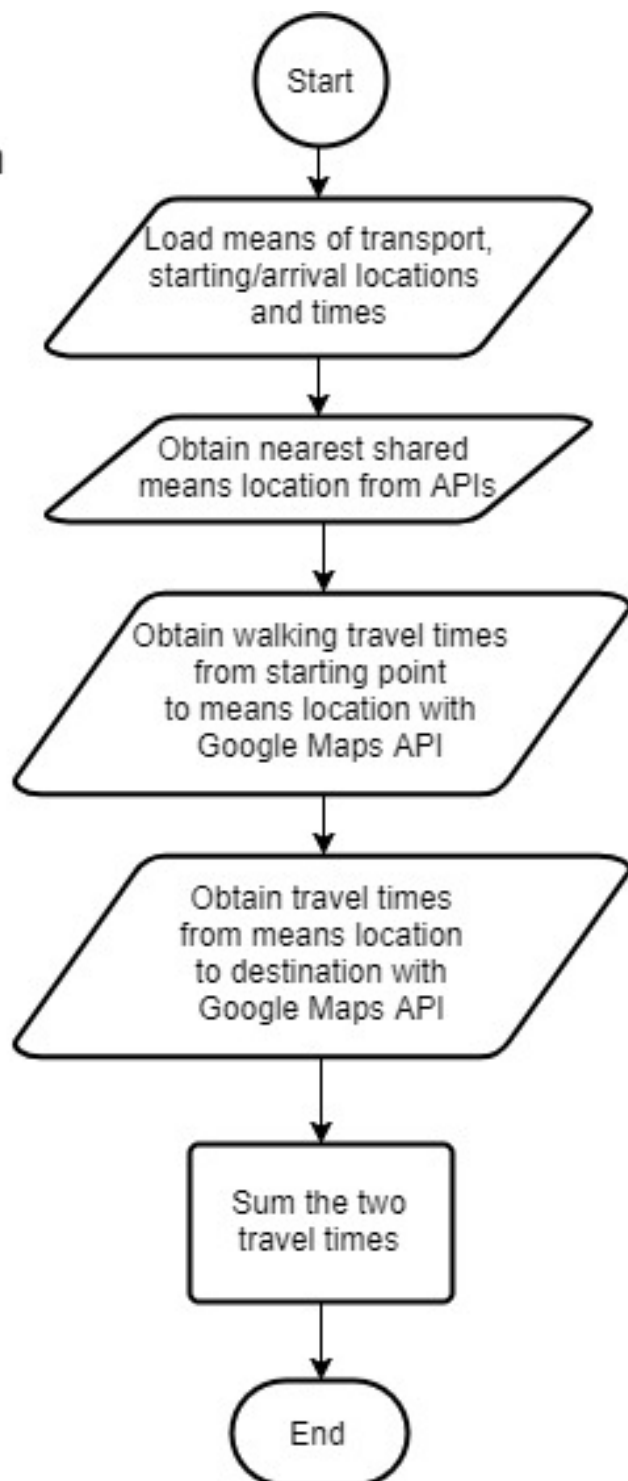


Figure 3.6: Sharing means of transport algorithm



**Dynamic events  
schedule algorithm**

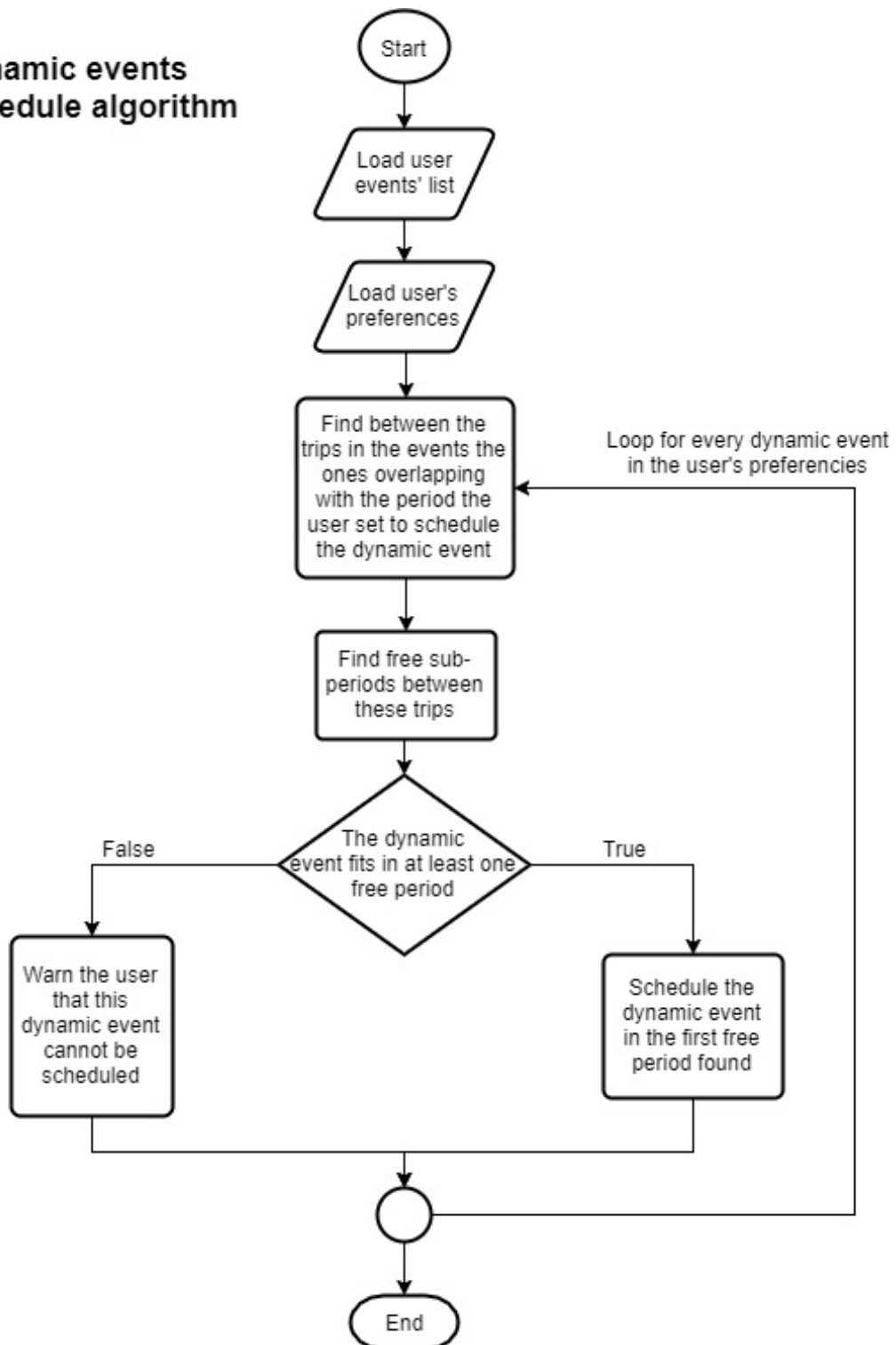


Figure 3.7: Dynamic event schedule algorithm

### 3. ALGORITHM DESIGN

---

```
1
2 public void dynamicScheduleAlgorithm() {
3     Iterator dEvents = user.getPreferences().getDynamicEventsList
4         ().iterator();
5     personalEventsController.resetDynamicEvents();
6     DynamicEvent d = dEvents.next();
7     for(Event e : personalEventsController.getEventsList()){
8         while(e.getFromTrip().getEndTime().isAfterThan(d.
9             getBeginningTimeOfPeriod())){
10            if(personalEventsController.getNext(e).getToTrip().isEqual
11                (e.getFromTrip())){
12                if(Time.getBetweenMinutes(personalEventsController.
13                    getNext(e).getStartTime(), e.getFromTrip().getEndTime
14                        ()) >= d.getBreakMinutes()){
15                    if(!personalEventsController.
16                        isOverlappingAnotherDynamicEvent(d)){
17                        personalEventsController.addDynamicEvent(d, e.
18                            getFromTrip().getEndTime());
19                        d = dEvents.next();
20                    } else {
21                        dynamicUnfeasibilityWarning(d);
22                    }
23                }
24            } else {
25                if(Time.getBetweenMinutes(personalEventsController.
26                    getNext(e).getToTrip().getStartTime(), e.getFromTrip
27                        ().getEndTime()) >= d.getBreakMinutes()){
28                    if(!personalEventsController.
29                        isOverlappingAnotherDynamicEvent(d)){
30                        personalEventsController.addDynamicEvent(d, e.
31                            getFromTrip().getEndTime());
32                        d = dEvents.next();
33                    } else {
34                        dynamicUnfeasibilityWarning(d);
35                    }
36                }
37            }
38        }
39    }
40 }
```

# Chapter 4

## User Interface Design

This section is a recapitulation of the section 3.B.1 (User Interfaces) of the Requirements Analysis and Specification Document and a deepening of design aspects of the user interface. The application will be developed as a mobile application for the main mobile operating systems (iOS and Android). As the system will appear the same for all the users, it will provide all the functionalities described in the RASD, in a unique user interface.

### 4.A Mockups

Following some mockups will provide an idea of the user interface while the user interacts with it, making use of the functionalities of the application. It includes most of the main screen that the user will face.

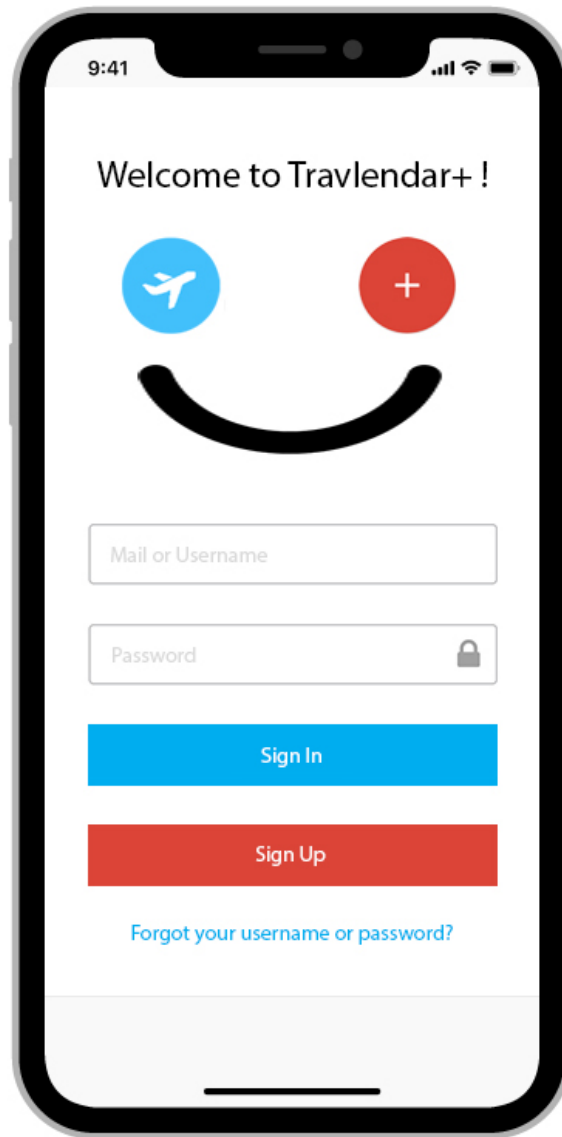
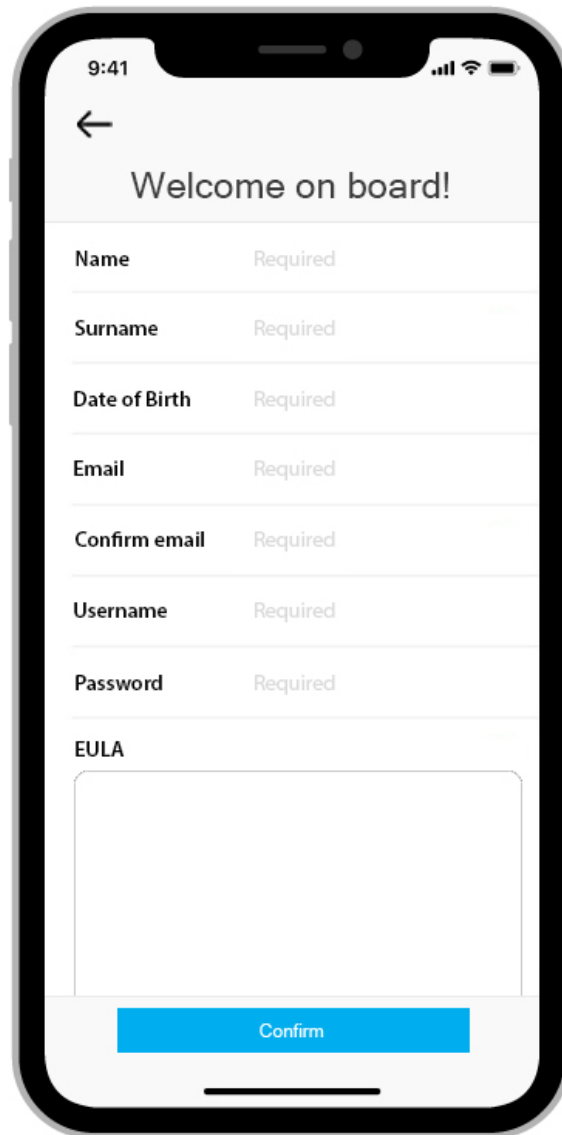


Figure 4.1: Mockup of the login screen



A mockup of a mobile registration screen. The screen is framed by a black border. At the top, the status bar shows the time 9:41, signal strength, Wi-Fi, and battery. Below the status bar is a white header with a back arrow on the left and the text "Welcome on board!" in the center. The main content area is a white form with several input fields, each with a label and a "Required" status. The fields are: Name, Surname, Date of Birth, Email, Confirm email, Username, and Password. Below these fields is a section labeled "EULA" with a large, empty rectangular box. At the bottom of the form is a blue button with the text "Confirm".

Field	Required
Name	Required
Surname	Required
Date of Birth	Required
Email	Required
Confirm email	Required
Username	Required
Password	Required

EULA

Confirm

Figure 4.2: Mockup of the registration screen

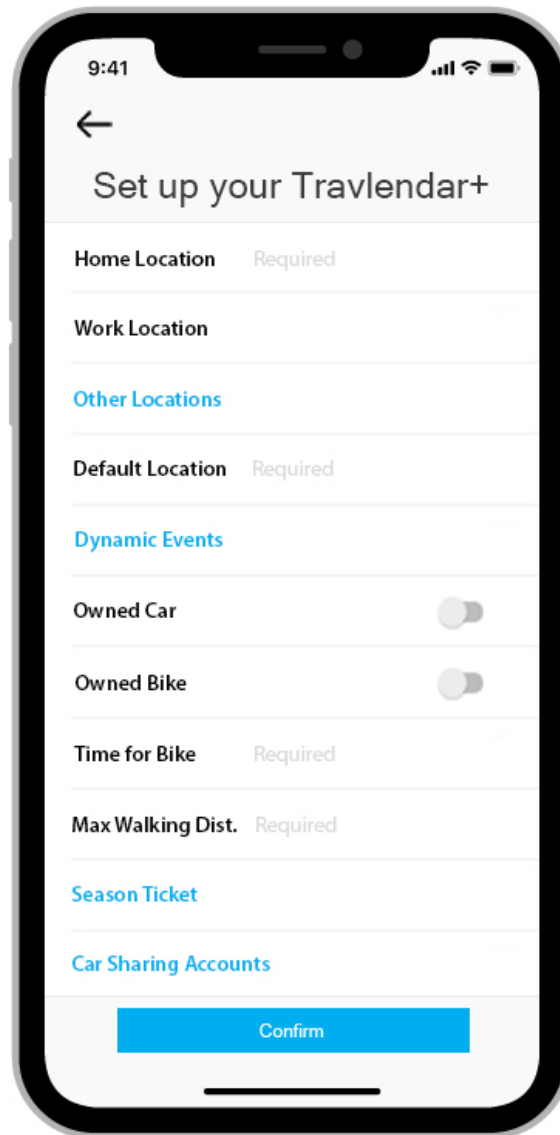


Figure 4.3: Mockup of the screen where the user can set his preferences

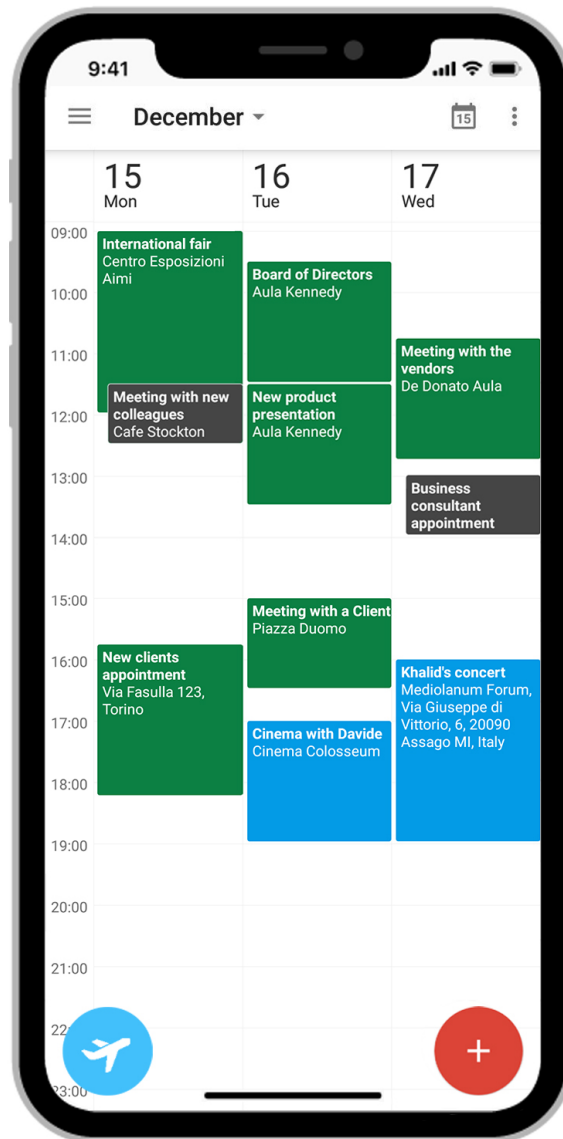


Figure 4.4: Mockup of the calendar screen with primary and secondary events

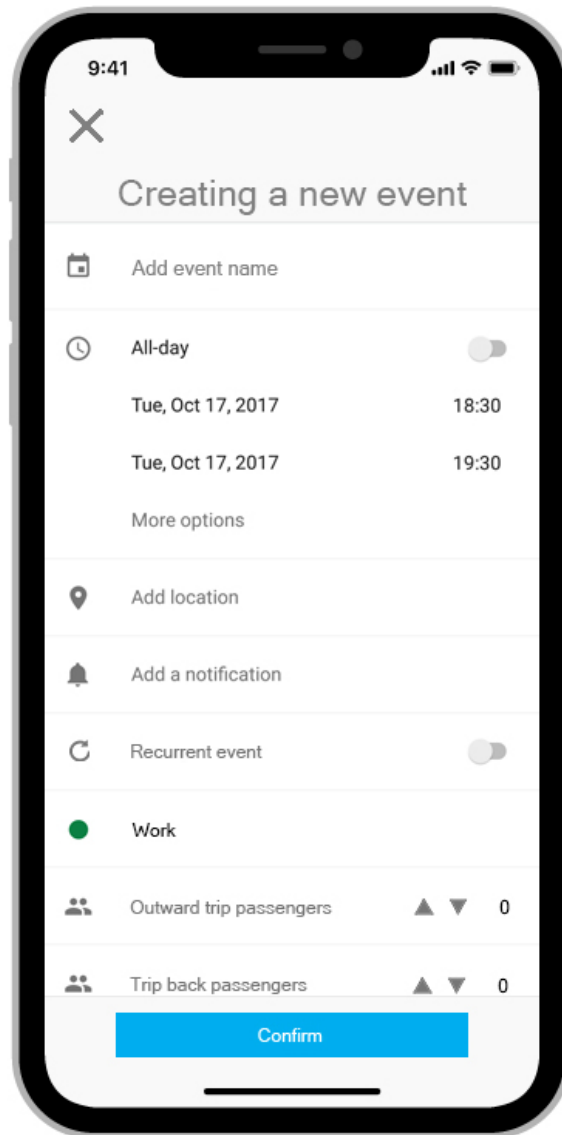


Figure 4.5: Mockup of the screen that allows to add an event





Figure 4.6: Mockup of the screen with the trips shown

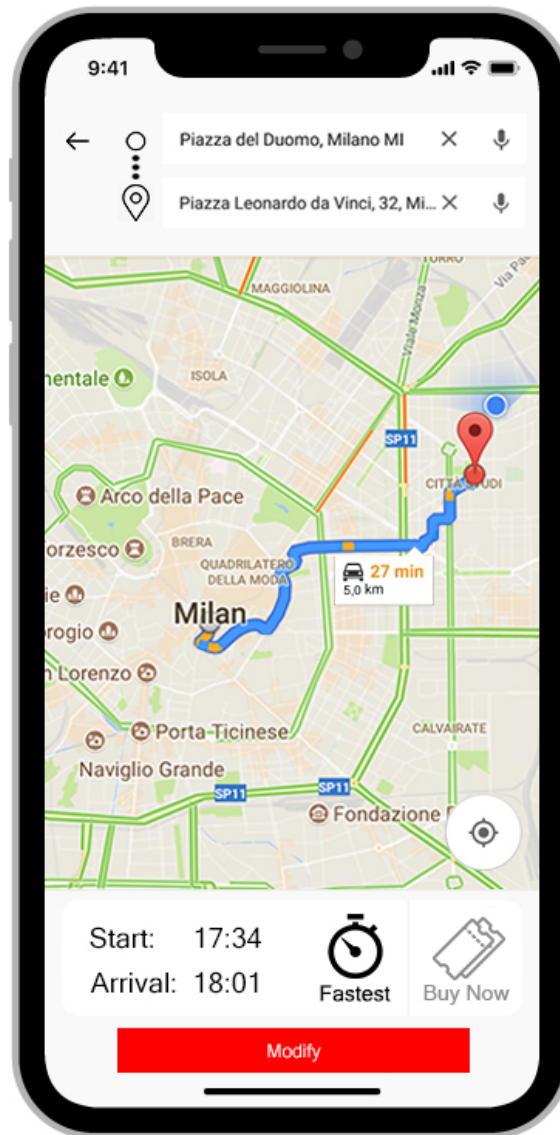


Figure 4.7: Mockup of the trip details screen

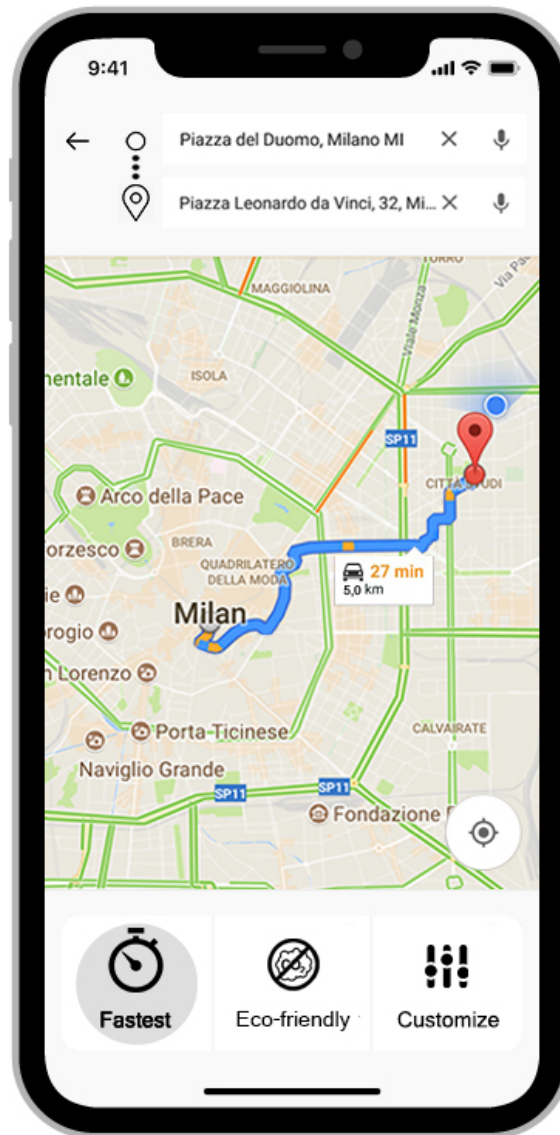


Figure 4.8: Mockup of the screen that allows the user to modify a trip

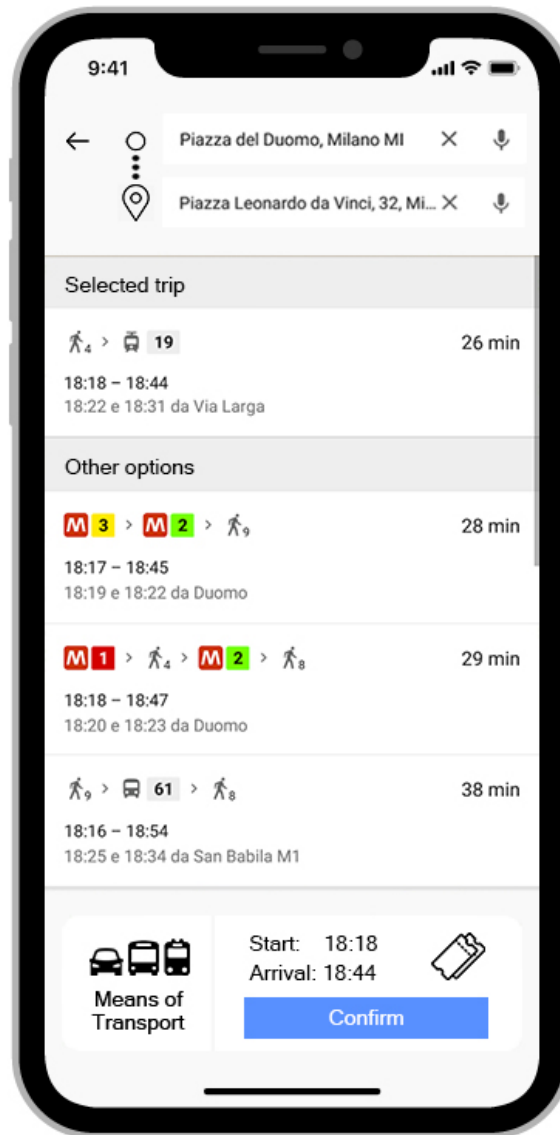


Figure 4.9: Mockup of the screen with the customization of a trip

## 4.B UX Diagrams

UX diagrams provide information about the user interface of the system and how the user interacts with it. For the diagram comprehension purposes, additional screens used to add specific information (other locations, dynamic

events, season tickets, etc...) in the preferences setup are not considered and different ways to buy tickets (opening browser or calling an external application) or reserve a sharing vehicle or bike are modeled as a single object.

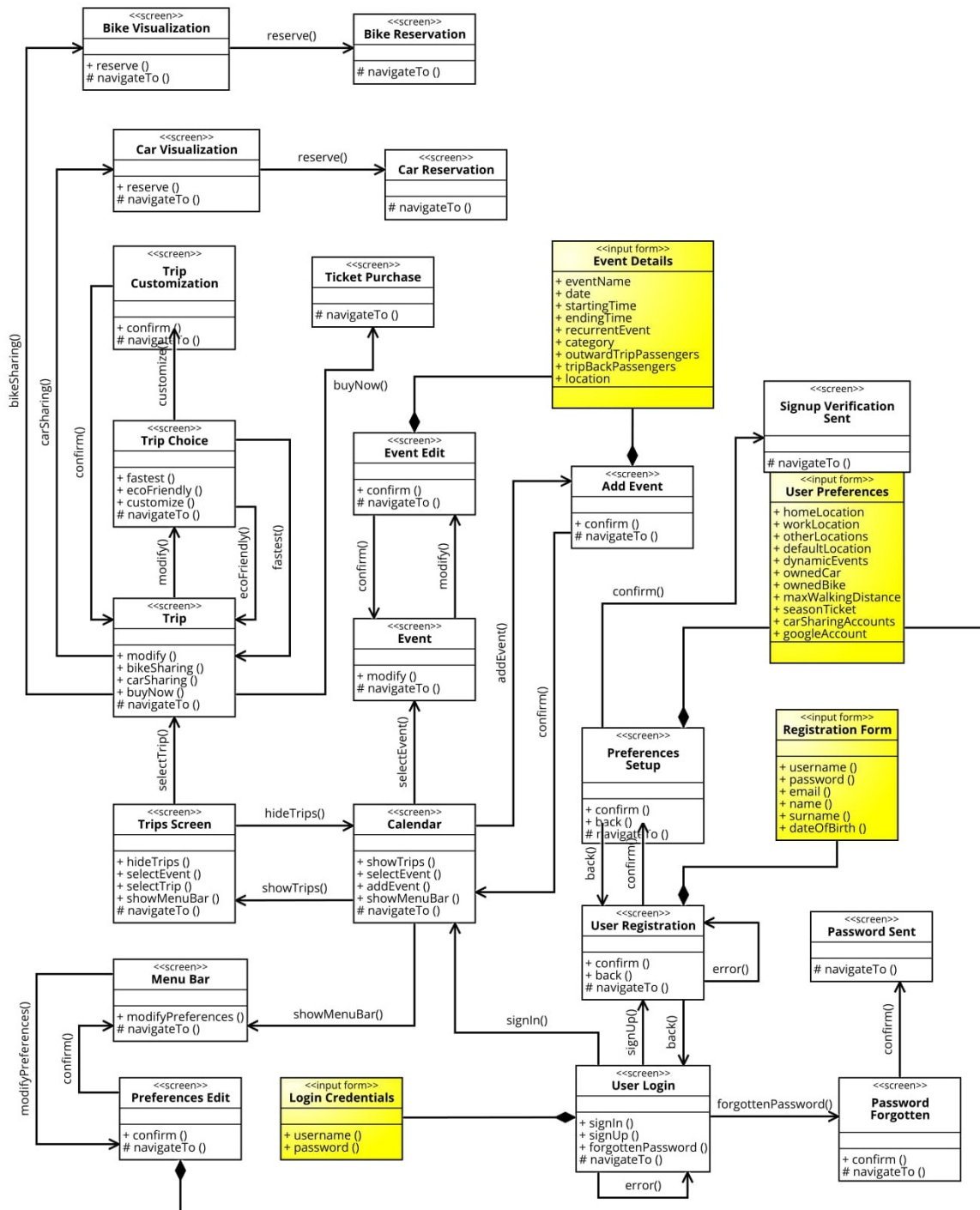


Figure 4.10: UX diagram of the application

## 4.C BCE Diagrams

For the implementation of the system, a Model-View-Controller design pattern is adopted and BCE diagrams are useful to show how user interactions are managed internally by the system. Boundaries are objects that interface with the users of the application; Entities object model the access to data; Controls object manage the communication between boundaries and entities.

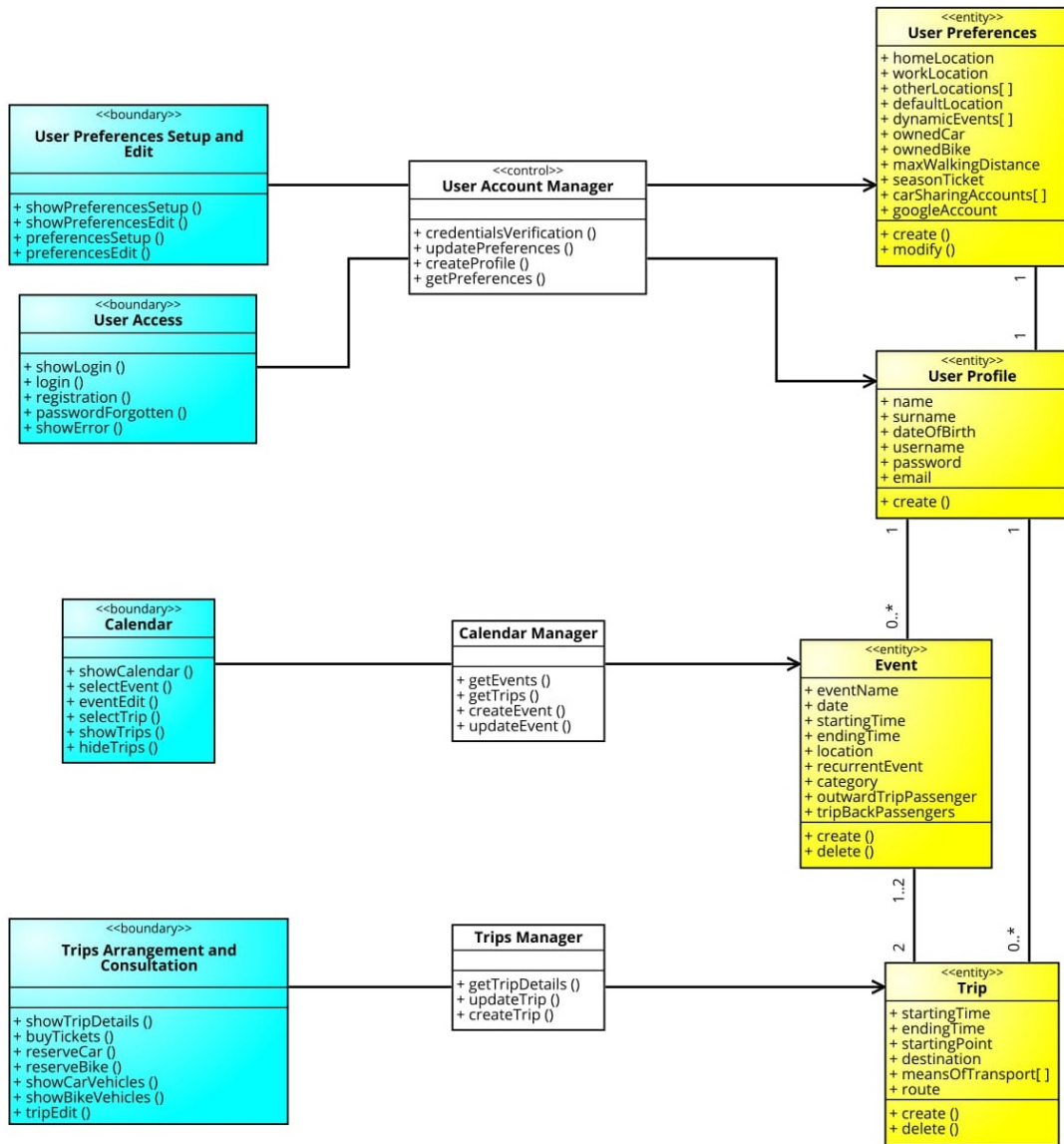


Figure 4.11: BCE diagram of the application

## Chapter 5

# Requirements Traceability



# Chapter 6

## Implementation, Integration and Test Plan

### 6.A Implementation Order

The implementation order of the tiers of the system is chosen in order to ease and speed up the integration of the complete system. The first step is to have the database tier and then we can develop the business tier because it can be tested without any client but only by making API calls and it is requested to use the mobile application. The next thing to be developed is the mobile application or the web tier, but the decision was to develop the mobile application so that a client-server system is obtained. Then in order the web tier and the web browser will be implemented.

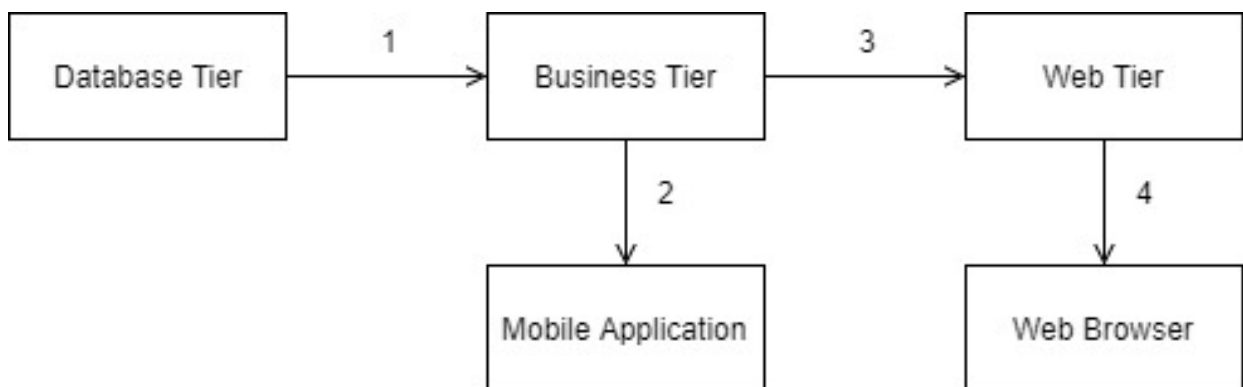


Figure 6.1: Implementation order of the system

## 6.B Integration and Test Plan

This section describes the preparation of the integration testing activity for all the components, to reach the completed system.

### 6.B.1 Entry criteria

In order to enter in the integration testing phase some condition have to be satisfied, first thing RASD and DD of the system should be delivered. Furthermore, the development percentage of the components involved in the activity of integration has to be over a certain value (80% for the components of this system) and the relative test units should be performed.

### 6.B.2 Elements to Be Integrated

Referring to the section 2 - Architectural design, the system can be divided in three categories of components:

- Front-end components: mobile application;
- Back-end components: the server and its components;
- External components: the components which refer to functionalities provided by external systems and the DBMS.

The front-end and the external components are independent one from each other, referring to back-end components some of them are not independent from others and need to be integrated. To fully integrate the three categories of components, some partial integration between categories need to be performed: front-end components and back-end components, and back-end components with external components.

### 6.B.3 Integration Strategy

The approach used for the integration testing phase is a bottom-up strategy, starting from components independent from other ones or components that depends only on one already developed. Then subsystem formed by the components tested, in turn, will be tested. This method will allow that a single component can be tested as soon as it's finished (or almost completed), and so to parallelize the two phase of development and testing.

# Chapter 7

## Effort Spent

Date	A. Aimi	R. Bigazzi	F. Collini
15/11/17		1,5h	1,5h
17/11/17		3h	4h
18/11/17	3h	3h	5h
19/10/17	2h		2h
20/10/17	2h	2h	2h
21/10/17	3h	1h	3h
22/10/17	6h	4h	
23/10/17	3h	2h	3h
24/10/17	4h	4h	4h
25/10/17	7h	7h	5h
26/10/17	6h	6h	6h
Tot	36h	33,5h	35,5h

Table 7.1: Hours of work spent by the authors of the document