Politecnico di Milano
AA 2017-2018

Computer Science and Engineering

Software Engineering 2 Project

Travlendar+

Design Document

Andrea Mafessoni - 899558
Andrea Mazzeo - 895579
Daniele Moltisanti - 898977

Deliverable:	DD
Title:	Design Document
Authors:	Andrea Mafessoni, Andrea Mazzeo, Daniele Moltisanti
Version:	1.0
Date:	26-November-2017
Download page:	https://github.com/AndreaMazzeo289/MafessoniMazzeoMoltisanti.git
Copyright:	Copyright © 2017, Andrea Mafessoni, Andrea Mazzeo, Daniele Moltisanti – All rights reserved

Contents

List of Figures

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

1.3.2 Acronyms

1.3.3 Abbreviations

1.4 Document structure

2 Architectural design

2.1 Overview: High-level components and their interaction

2.2 Component view

2.3 Deployment view

2.4 Runtime view

2.5 Component interfaces

2.6 Selected architectural styles and patterns

3 Algorithm design

In the following paragraph are presented the most relevant and significant algorithm used in Travlendar+ application. More specifically are described the following algorithm:

- Compute travel;
- View daily schedule;
- Check appointments overlapping;
- Check appointments unreachability;
- Check travel alternatives;
- Check movement alternatives;

The application is written in Java, but the algorithms are shown in pseudocode.

3.1 Object class

Before to explain the algorithms details, it is needed to introduce the class Appointment, Travel and Movement that are the most important entities for the whole application.

3.1.1 Appointment class

The class Appointment has as attributes:

- The appointment name;
- The date;
- The time when it starts;
- Address of destination;
- Address of departure;
- The desired time to leave;
- The expected time of arrival;
- The appointment duration;
- The associated travel to reach the appointment;

Here the appointment class written in Java:

```
public class Appointment {

        String name;
        Date date;
        Time beginTime;
        Address destination;
        Address departure;
        Time departureTime;
        float arrivalTime;
        float duration;
        Travel travel;
}
```

3.1.2 Travel class

The travel class has the following attributes:

- The related appointment;
- The departure point;
- The destination point;
- The travel type;
- The weather conditions;
- The movements list in which the travel is split;

Here the travel class written in Java:

```java
public class Travel {

        Appointment appointment;
        Address departure;
        Address destination;
        TravelType travelType;
        Weather weatherConditions;
        ArrayList<Movement> movements;
}
```

3.1.3 Movement class

The movement class has the following attributes:

- The departure point;
- The destination point;
- The estimated time;
- The time of departure;
- The movement type;

Here the movement class written in Java:

```java
public class Movement {

        Address departure;
        Address destination;
        TravelType travelType;
        Time estimatedTime;
        Time departureTime;
}
```

3.2 Algorithms

3.2.1 Compute travel

The function computeTravel is used to create a new travel for a specific appointment. The followed procedure is composed by four steps:

1. Read the different information from the appointment passed as parameters.
2. Call the function queryMaps. this uses Google Maps API to compute the travel, the response is JSON object and the function parse this and extracts all information related to the computed travel.
3. For each steps, which put together with the other steps produces the travel, is created a new Movement and it is added to MovementList presents in Travel class.
4. Call the function computeWeatherCondition, that with specific API can compute the weather condition, and if the travel includes some walking or bicycling movement and it is expected 'rain', the system shows to the user a message.

```
function COMPUTETRAVEL(appoinment)
    READ(appointment.departure)
    READ(appointment.destination)
    READ(appointment.date)
    READ(appointment.beginTime)
    READ(appointment.travelPreferences)
    travel = QUERYMAPS(departure,destination,travelPreferences)
    for all movement detected in travel do
        createdMovement = CREATEMOVEMENT(movement)
        ADDTOMOVEMENTLIST(createdMovement)
    end for
     weather = COMPUTEWEATHERCONDITION(departure,destination, date, time)
    if wheater is 'rain' and ( (travelPreferences is 'green') or (exists one movement : movementType
is 'walk' or 'bike') ) then
        NOTIFYUSER("Rain expected, not reccomended use of bike or walks")
    end if
end function
```

3.2.2 View daily schedule

When the user clicks on view daily schedule button, the system computes the daily schedule through this function and show to the user all appointments expected for the selected day and all travel to reach them. This method has only one parameter, the day desired to compute the schedule.
The function is composed by four steps:

1. Extract first appointment expected in the selected day (appointment-i) and check its unreachability.
2. Extract the second appointment (appointment-i+1) and check its unreachability.
3. If both appointments are reachable then it is necessary to check the overlapping between them.
4. If all controls are successfully passed then it is possible to compute the travel for the appointment-i.
5. Go on with the next appointment and repeats the loop until all appointments are examinated.

```
function VIEWDAILYSCHEDULE(day)
    for all appointment in day do
```

```
            if CHECKUNREACHABILITY(appointment-i) is unreachable then
                MANAGEUNREACHABILITY(appointment-i)
            end if
        end for
        if CHECKUNREACHABILITY(appointment-i+1) is unreachable then
            MANAGEUNREACHABILITY(appointment-i+1)
        else if CHECKOVERLAP(appointment-i, appointment-i+1) is overlap then
            MANAGEOVERLAP(appointment-i,appointment-i+1)
        end ifCOMPUTETRAVEL(appointment-i) INCREMENT(i)
    end function
```

3.2.3 Check overlap

When the system has to check the overlapping between two appointments invokes this function that has as parameter the two appointments. The function controls which appointment starts before and then controls if the begin time of the second appointment overlaps with end time of the first appointment.

```
    function CHECKOVERLAP(app1, app2)
        if app1.beginTime is before of app2.beginTime and (app2.beginTime is before (app1.beginTime +
    app1.duration)) then
            return overlap
        else if app1.beginTime is before (app2.beginTime + app2.duration) then
            return overlap
        end if;
        return no overlap
    end function
```

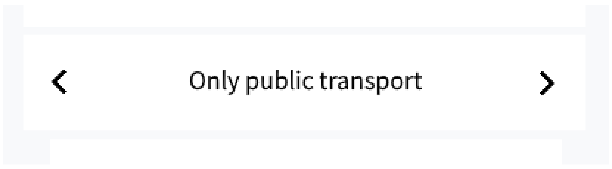
3.2.4 Check unreachability

This function checks the unreachability of one appointment passed as parameter. To check the unreachability it is necessary to control if the arrival time of one appointment is compatible with departure time added to estimated travel time.

```
    function CHECKUNREACHABILITY(appointment)
        if appointment.arrivalTime is before (appointment.departureTime + travel duration) then
            return unreachable;
        else
            return reachable;
        end if
    end function
```

3.2.5 Check travel alternative

The user can check all available travel alternatives and modify the actual one with another. This is possible switching the different alternatives.



The function checkTravelAlternative reads the selected alternative and computes the travel with the in-

serted TravelType.
View more detailed these steps:

1. There is a control on a travel type selected from the user and there are two possible cases:
   (a) 'only-own-car' or 'only-public-transport' or 'green: when one these travel type is selected, it is created a new travel alternative based on related travel mode (driving for only-own-car, transit for only-public-transport and bicycling for green).
   (b) 'faster' or 'cheaper': in this case, it is created a set that contains all possible travel computable.
2. Then it is selected the travel alternative:
   (a) In case of cheaper or faster travel, it is necessary to find among all computed travels the cheaper or the faster.
   (b) In other cases, return the only one alternative computed.

```
function CHECKTRAVELALTERNATIVE(travel, travelType)
    if travelType is 'only-own-car' then
        compute driving travel alternative
    else if travelType is 'only-public-transport' then
        compute transit travel alternative
    else if travelType is 'green' then
        compute bicycling travel alternative
    else if travelType is 'faster' or travelType is 'cheaper' then
        compute a set of differents travel alternatives
    end if
    if travel alternative exists then
        if travelType is 'faster' then
            find faster travel in the computed set and return it
        end if
        if t thenravelType is 'cheaper'
            find cheaper travel in the computed set and return it
        end if
        return the travel alternative found
    end if
end function
```

3.2.6 Check movement alternative

```
function CHECKMOVEMENTALTERNATIVE(movement, movementType)
    if movementType is 'car' then
        compute driving movement
    else if movementType is 'walk' then
        compute walking movement
    else if movementType is 'public-transport' then
        compute transit movement
    else if movementType is 'bike' then
        compute bicycling movement
    else if movementType is 'car-sharing' then
        compute walking movement to reach the car
        add a further driving movement
    **else if** movementType is 'bike-sharing' **then**
        compute walking movement to reach the bike
        add a further bicycling movement
    **end if**
    **if** computed movement exists **then**
        **return** movement alternative
    **end if**
**end function**
```

4 User interface design

4.1 Registration/Login

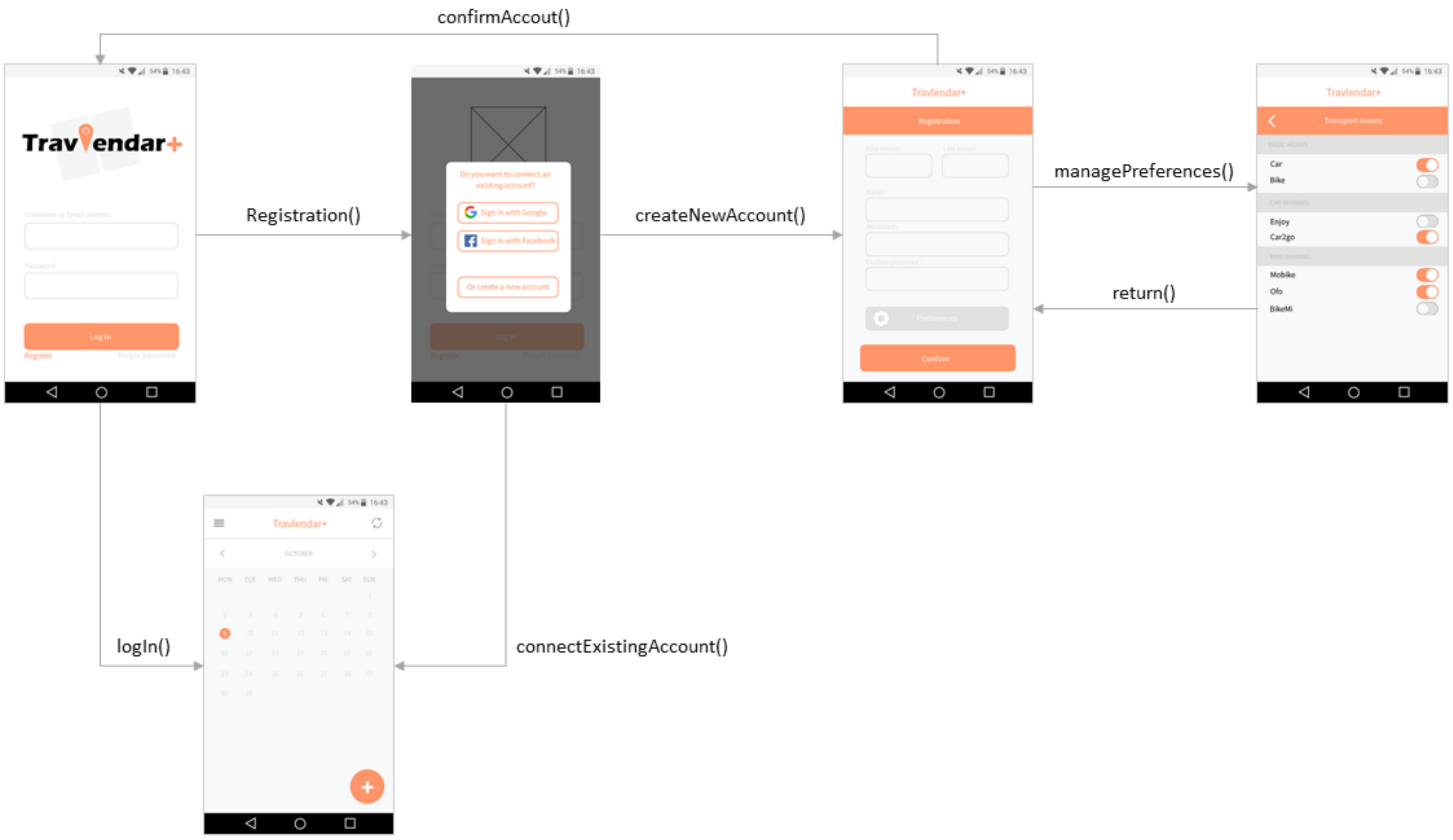


Figure 1: User Interface: registration and login

4.2 Calendar views

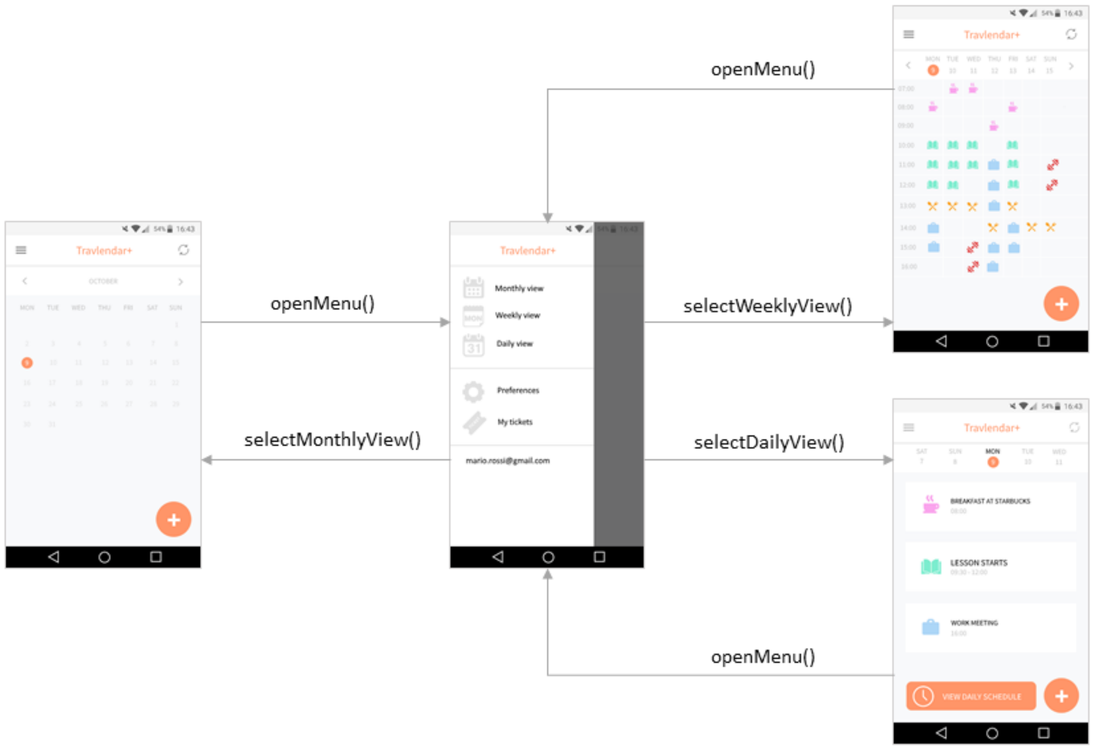


Figure 2: User Interface: calendar views and menu

4.3 Create new appointment

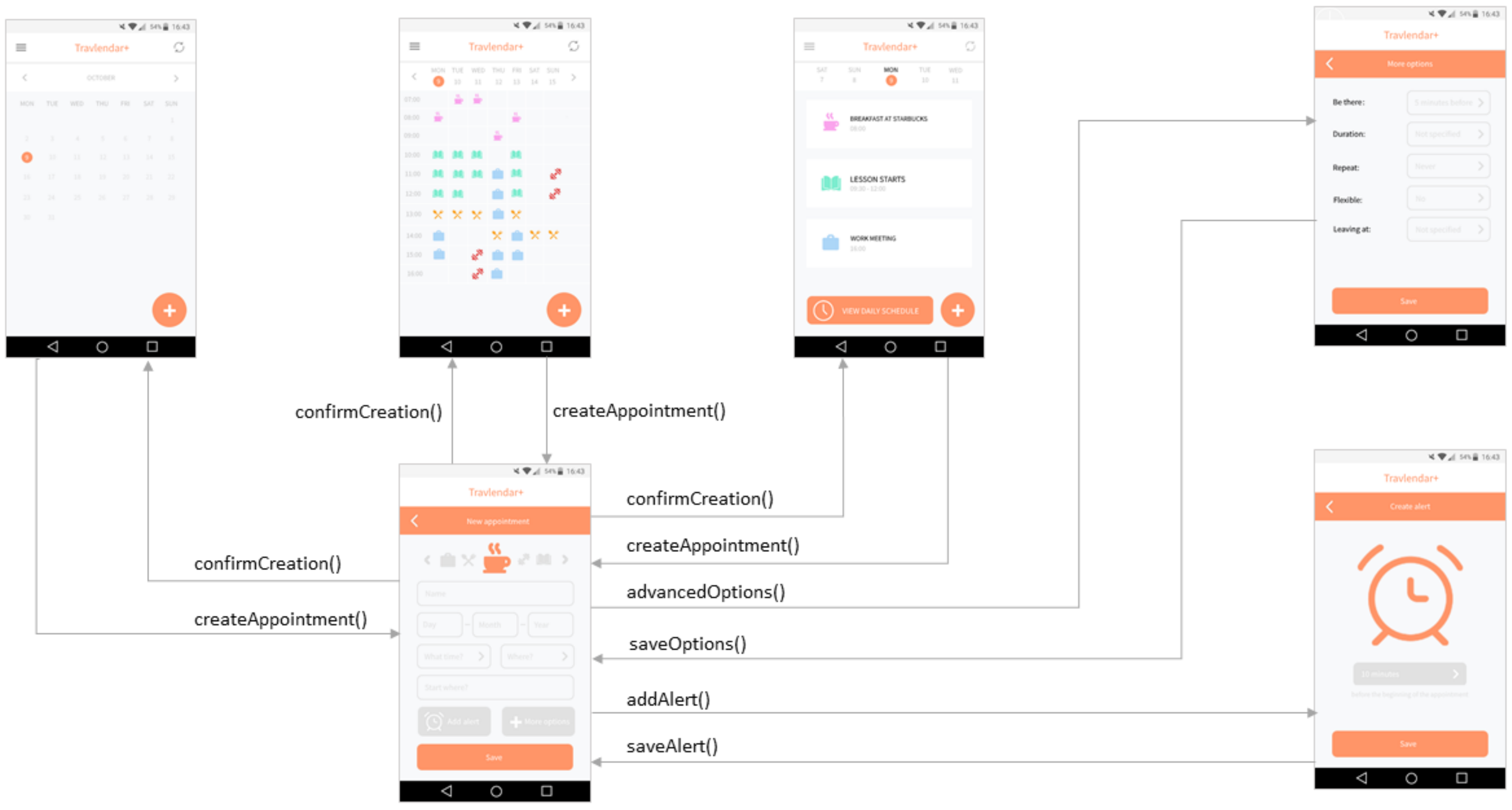


Figure 3: User Interface: creation of new appointment

4.4 Edit existing appointment

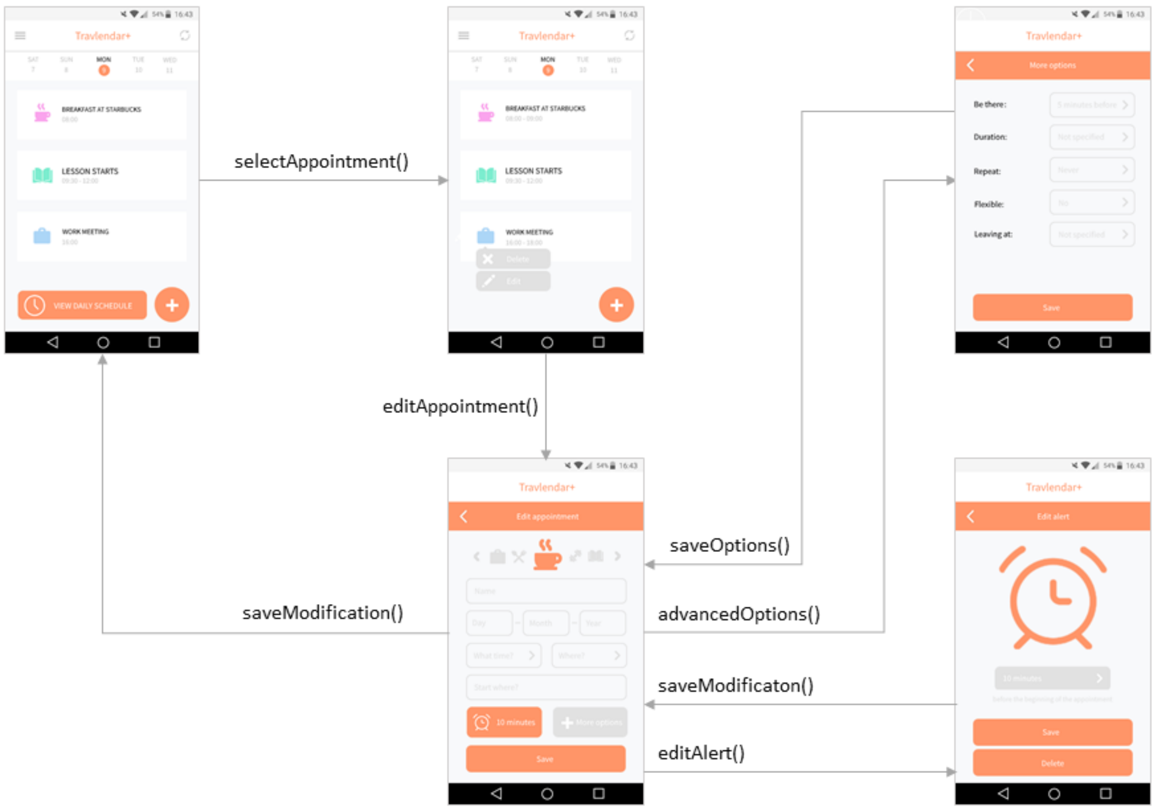


Figure 4: User Interface: edit of an existing appointment

4.5 manage preferences

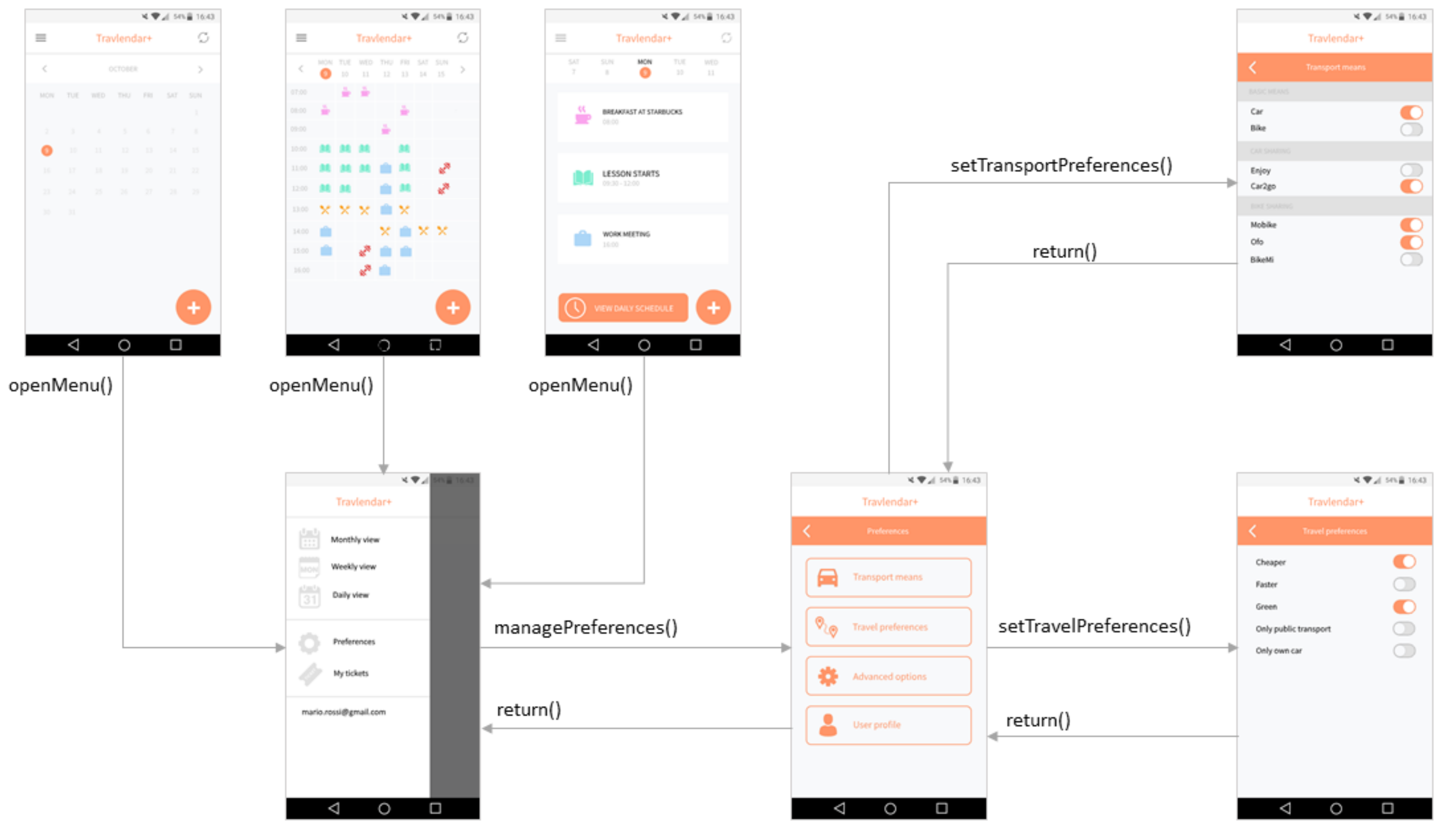


Figure 5: User Interface: manage preferences

4.6 View daily schedule and travel/movement details

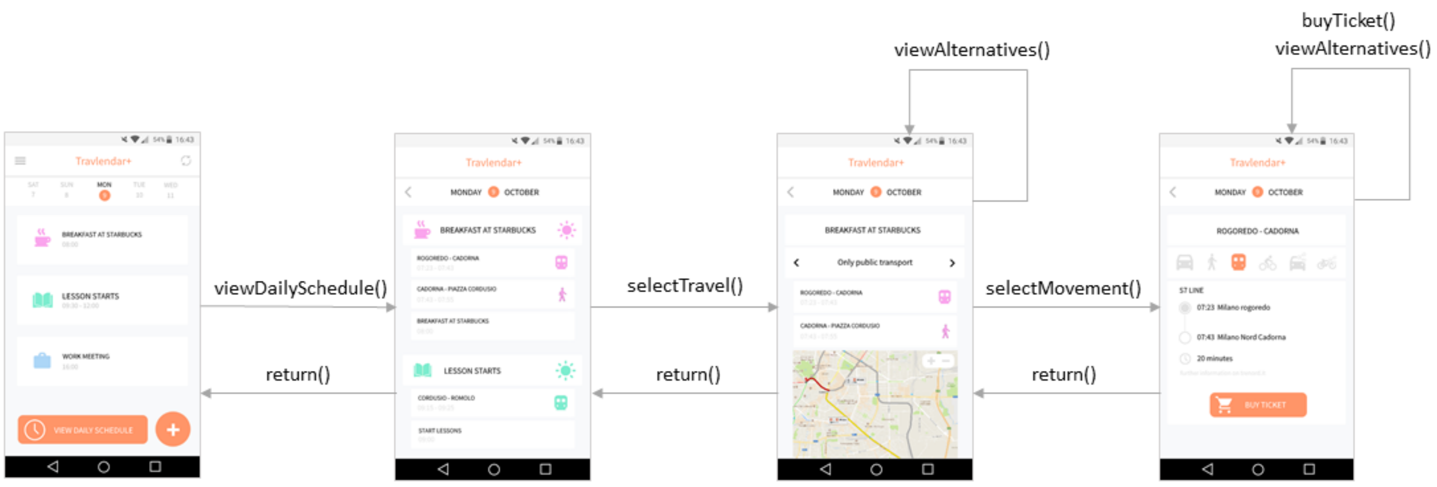


Figure 6: User Interface: daily schedule and travel/movement details

4.7 Buy and control bought tickets

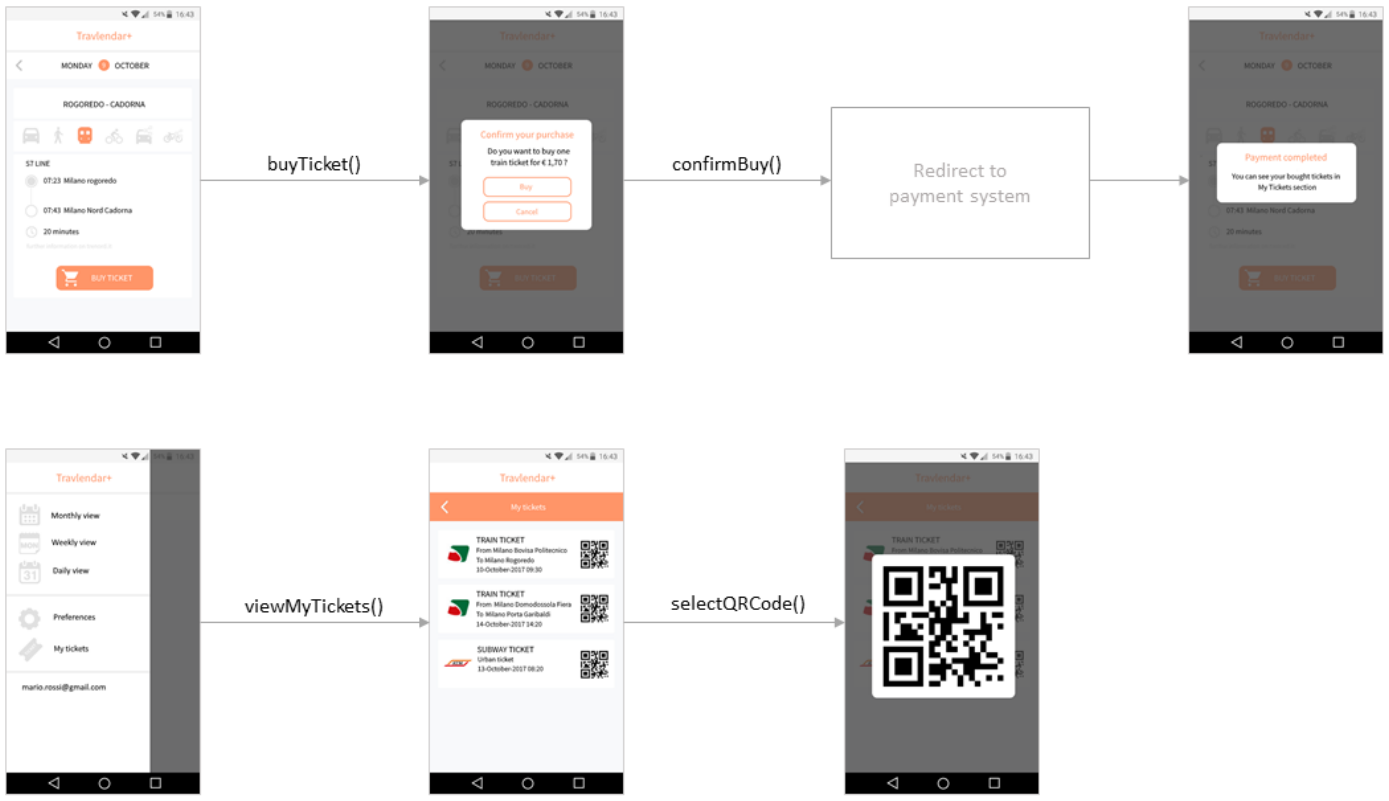


Figure 7: User Interface: buy ticket and 'My tickets' section

5 Requirements traceability

6 Implementation, integration and test plan

7 Effort spent