**Software Engineering 2: Travlendar+**

**D**esign **d**ocument

Agostini Andrea, Ciampiconi Lorenzo, Es-skidri Rachid

November 20, 2017

CONTENTS

[**INTRODUCTION** 3](#_Toc499026998)

[Purpose 3](#_Toc499026999)

[Scope 3](#_Toc499027000)

[Definitions 4](#_Toc499027001)

[Acronyms 4](#_Toc499027002)

[Abbreviations 4](#_Toc499027003)

[Reference Documents 5](#_Toc499027004)

[Document Structure 5](#_Toc499027005)

[**ARCHITECTURAL DESIGN** 6](#_Toc499027006)

[Overview 6](#_Toc499027007)

[Component View 6](#_Toc499027008)

[Deployment View 6](#_Toc499027009)

[RuntimeView 6](#_Toc499027010)

[Component Interfaces 6](#_Toc499027011)

[Selected Architectural Styles and Patterns 6](#_Toc499027012)

[Other Design Decision 6](#_Toc499027013)

[**ALGORITHM DESIGN** 7](#_Toc499027014)

[**USER INTERFACE DESIGN** 8](#_Toc499027015)

[Requirments Traceability 14](#_Toc499027016)

[**IMPLEMENTATION, INTEGRATION AND TEST PLAN** 15](#_Toc499027017)

[**EFFORT SPENT** 15](#_Toc499027018)

[**REFERENCES** 16](#_Toc499027019)

# INTRODUCTION

## 

## purpose

This is the Design Document for Travlendar plus system. Its aim is to provide a functional description of the main architectural components, their interfaces and their interactions, together with the algorithms to implement and the User Interface design. Using UML standards, this document will show the structure of the system and the relationships between the modules. It describes also information about the Test Plan.

This document is written for project managers, developers, testers and Quality Assurance. It can be used for a structural overview to help maintenance and further development.

## Scope

**Analysis of the world phenomena**

Nowadays, time management is one of the most important things in the context of today's society, especially in big cities, where the variety of means of transport is so great to allow a better optimization of a person's events organization. ​​Travlendar+ borns as  a digital calendar that allows the user not only to display his events, but also to provide the user the best way to reach the events in the best possible way, according to the criteria chosen by him.

**Analysis of the shared phenomena**

There are two different kinds of shared phenomena, the ﬁrst one  includes phenomena that are controlled by the world and observed by the machine, such as the GPS position of the user, the traffic, the weather and for example, something quite abstract such as the time schedule of a bus or a tube. The second one contains all those phenomena controlled by the machine and observed by the world (in according to the domain) such as : the user follows Travlendar+  indications, the user inserts the true duration time events, the user uses the system inside the correct geographic area etc.

## Definitions

* **User**: any individual subscribed to the service.
* **Visitor**: an individual not subscribed to the service.
* **Event**: an appointment that could be registered in the calendar.
* **Free Time Interval**: with this term we refer to the interval of time that user could registered in the calendar to indicate where the flexible break must be spent.
* **Overlapping events**: when two events A and B overlaps it means that they share a time interval. More formally, when A starts before the start of B and A ends after the end of B, A overlaps with B.
* **System**: the whole software system to be developed, comprehensive of all its parts and modules.

## Acronyms

* **RASD**: Requirements Analysis and Speciﬁcation Document (this document).
* **API**: Application Programming Interface.
* **UI**: User Interface.
* **DB**: Data Base.
* **DBMS**: Data Base Management System.
* **DD**: Design Document.
* **MVC**: Model View Controller.

## Abbreviations

* **[Gn]**: nth goal.

## Reference Documents

• This document refers to the specification document: Mandatory Project Assignments.pdf - Assignments AA 2017-2018

• This document refers to the RASD – the previous deliverable.

• This document refers to the GOF Design Patterns book.

.

## Document Structure

This document is structured in six parts:

**Chapter 1**: *Introduction*. It provides an overall description of the system scope and purpose, together with some information on this document.

**Chapter 2:** *Architectural Design*. This section shows the main components of the systems with their sub-components and their relationships, along with their static and dynamic design. This section will also focus on design choices, styles, patterns and paradigms.

**Chapter 3**: *Algorithm Design*. This section will present and discuss in detail the algorithms designed for the system functionalities, independently from their concrete implementation.

**Chapter 4**: *User Interface Design.* This section shows how the user interface will look like and behave, by means of concept graphics and UX modeling*.*

**Chapter 5**: *Requirements Traceability.* This section shows how the requirements in the RASD are satisﬁed by the design choices of the DD.

**Chapter 6**: *Implementation, Integration and Test plan.*

# ARCHITECTURAL DESIGN

## 

## Overview

**High level components and their interaction:**

*Database* :

the data layer is responsible for the data storage and retrieval.

It does not implement any application logic. This layer must guarantee ACID properties.

*Application* server :

this layer contains all the application logic of the system.

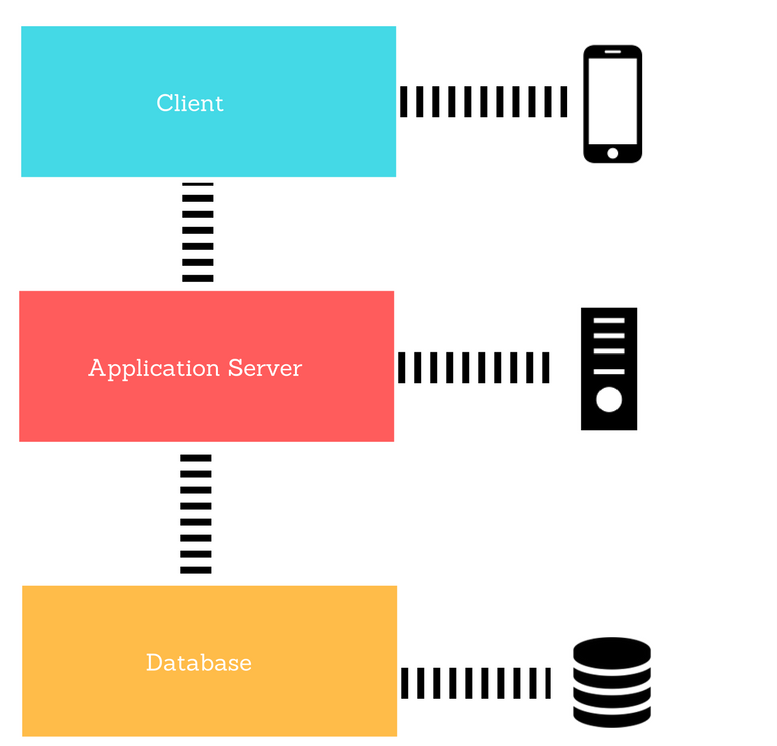
All the policies, the algorithms and the computation are performed

here. This layer offers a service-oriented interfaces.

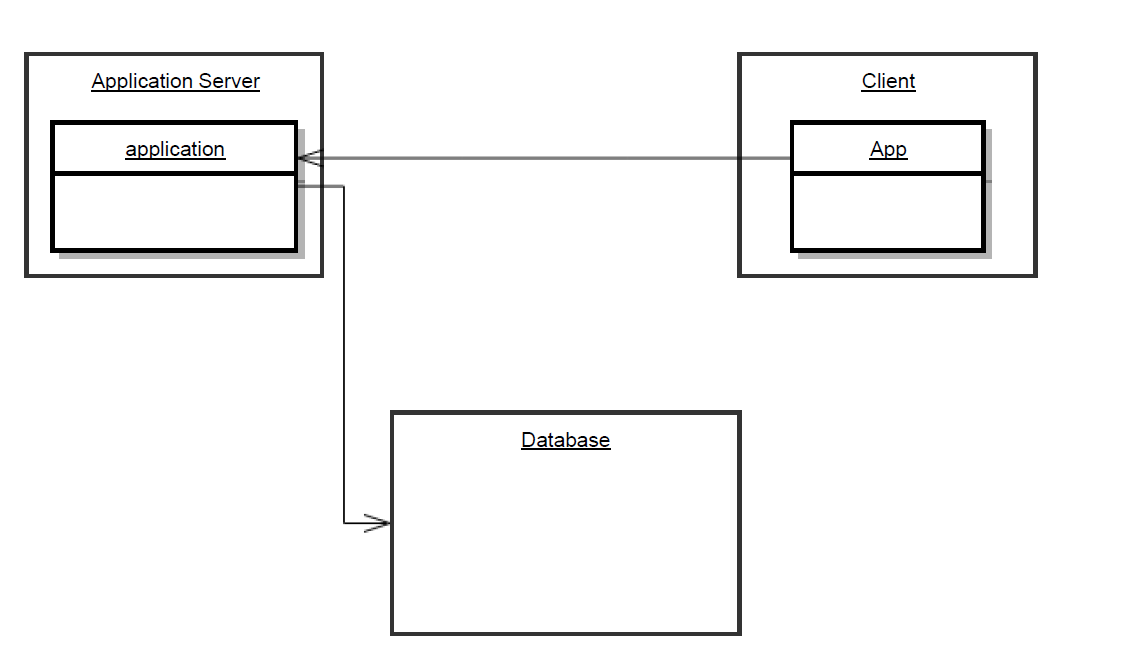
*Client :*

This layer consists in an Android Mobile application. It’s both presentation layer and logic/client layer, it communicates directly with the application server and it represents the user’s interface.

These high-level components are structured into three layers, shown in *figure 1*. This choice give us the possibility to compute all the business logic in the Application Server layer and make the client application thin and efficient so it could provide a comfortable user experience. Furthermore this design allows to extend the system, inserting a Web Server layer to consult Travlendar in every device that has a generic browser. In this first release we focus on the Android Mobile client in order to implement the system in the expected times.



*Figure 1: Layers of the system.*



*Figure 2: High level components of the system.*

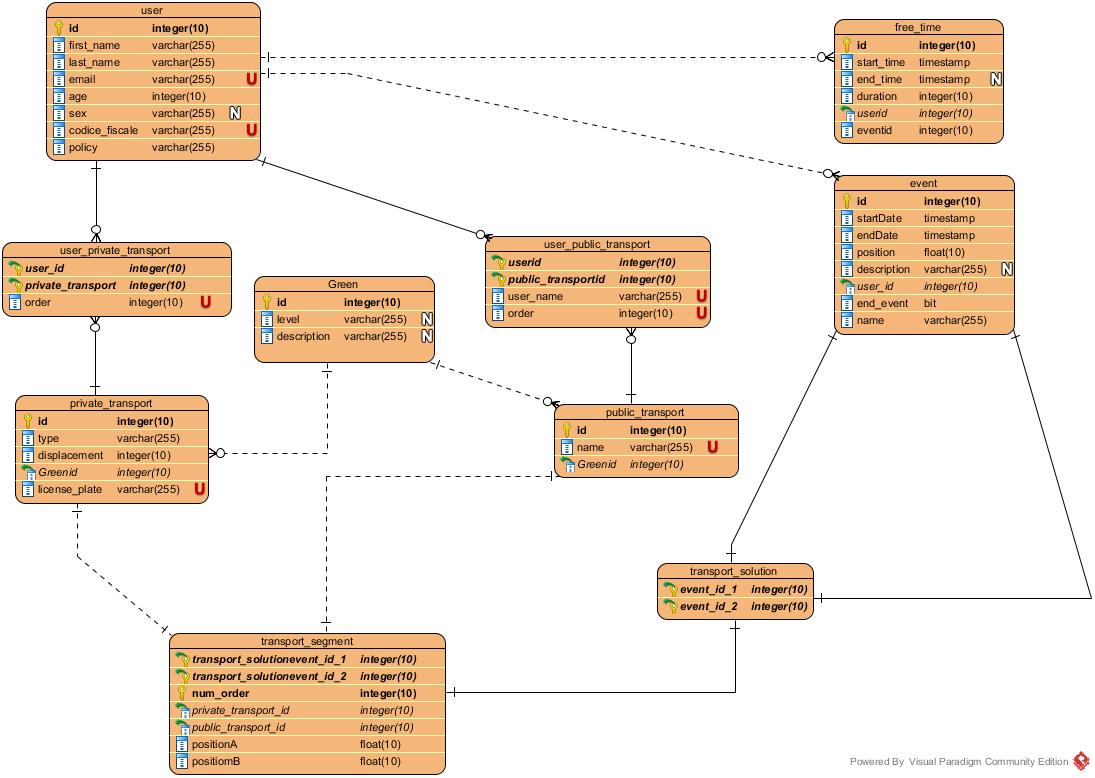
## Component View

The following diagrams describe the main component of the system and the interfaces through which they interact. The *client side* is identified with the mobile application. Is a very thin client that allows the user to interact with the application server which contains the main part of the business logic of the system.

**Server Side Database**

The database tier runs in an external database service that allow us to store data more safely than in an internal Db. We use InnoDB as the database engine: the DBMS has to support transactions and ensure ACID properties. Access to the data must be granted only to authorized users possessing the right credentials. Every software component that needs to access the DBMS.

It’ important to remark that the user entity in the following schema is useful for the business tier(for example, we can store here the custom user policies). The user credentials are handled by an external backend service, for high security reasons.

****

*Figure 3: The Entity-Relationship diagram of the database schema.*

**Client Side Database**

The Client Db schema is composed by a synchronized light copy of the Server side Db.

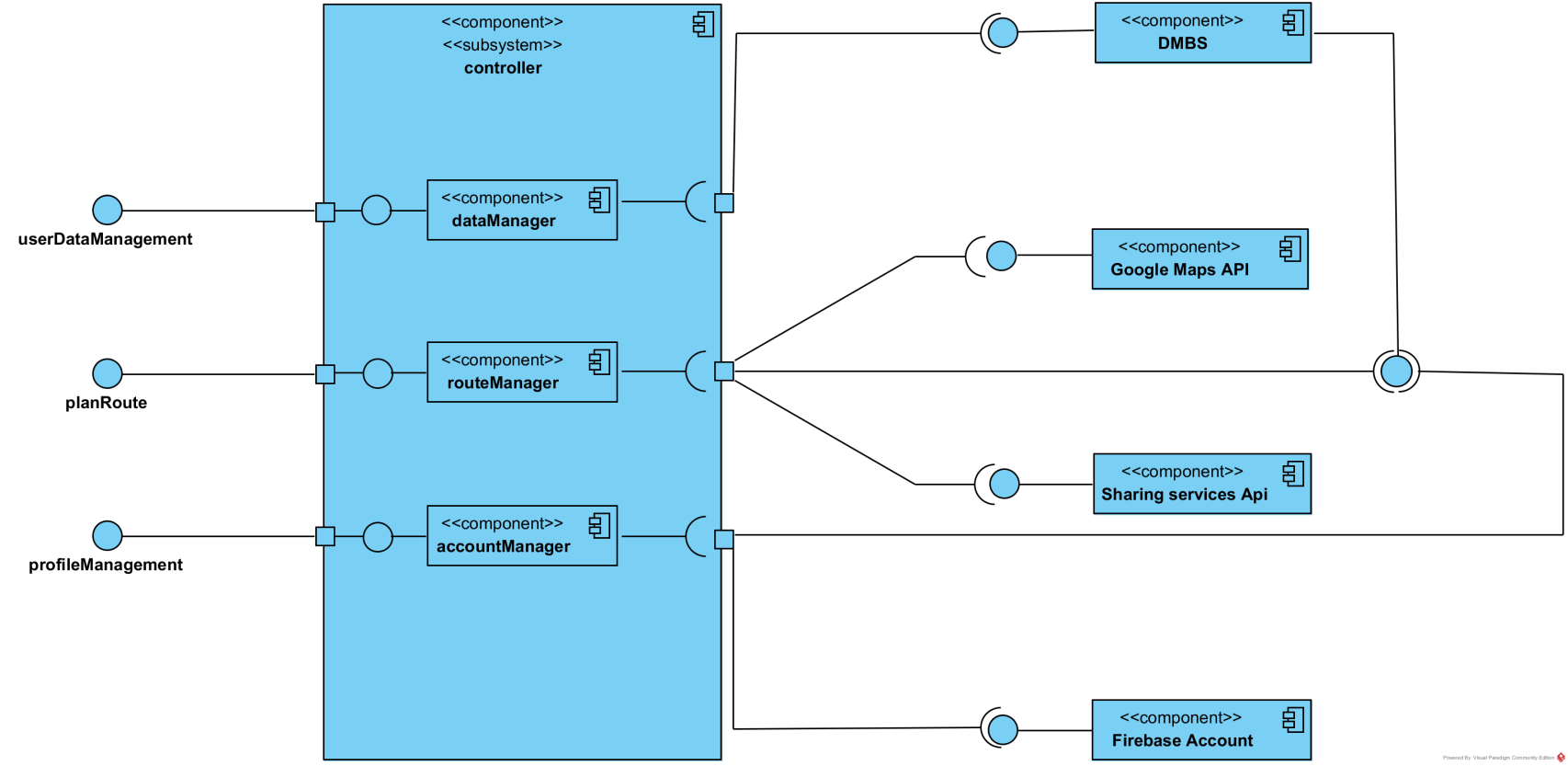
We chose SQLite DBMS because is a best practice implement that on an Android application. The interaction with that is handled by an Android ORM (Object Relational Mapping), in particular we use greeDAO ORM service, like in the *Figure 4* below.



*Figure 4: greenDao module connection*

**Controller projection**

Controller subsystem is composed by three main components: *DataManager, AccountManager* and *RouteManager.* The first one concerns all the business logic about the management of the data that will be putted/getted into/from the database. The RouteManager provides the logic about find the best solutions to link the events of the use with se Google Maps Api and Sharing services Api support . The accountManager has the goal to identify the user with the support of Firebase Api.

******

*Figure 5: Controller Projection Schema*

## Deployment View

The system architecture is divided in three tiers and it is based.

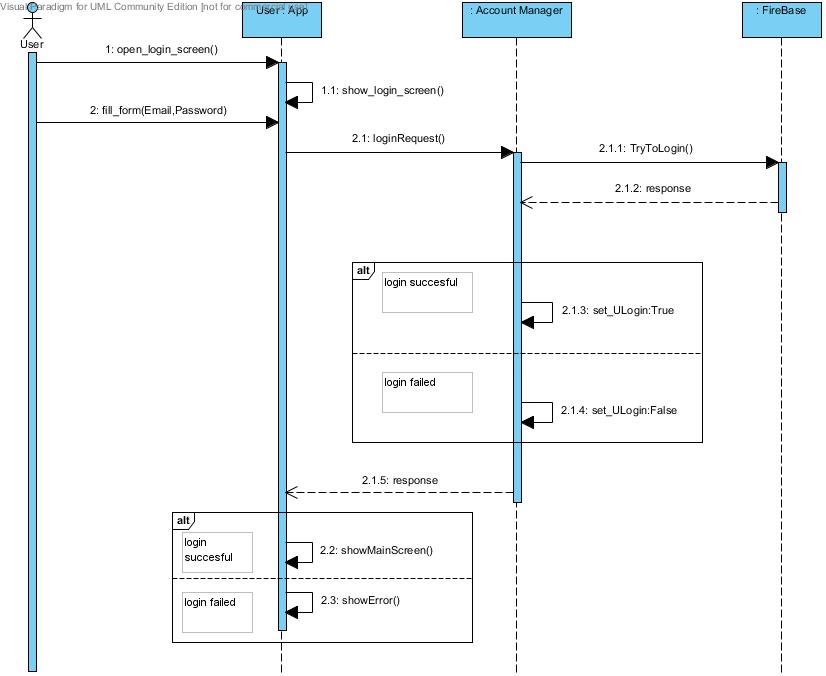
* The client tier is composed by the mobile application that communicate with the business tier
* The business tier is based on Spring framework because it represent the best practice for these types of system. The details of this framework are discussed in the next page.
* The Database tier is mainly composed by the External Database Server. The communication with the business tier is performed via JDBC connector.

****

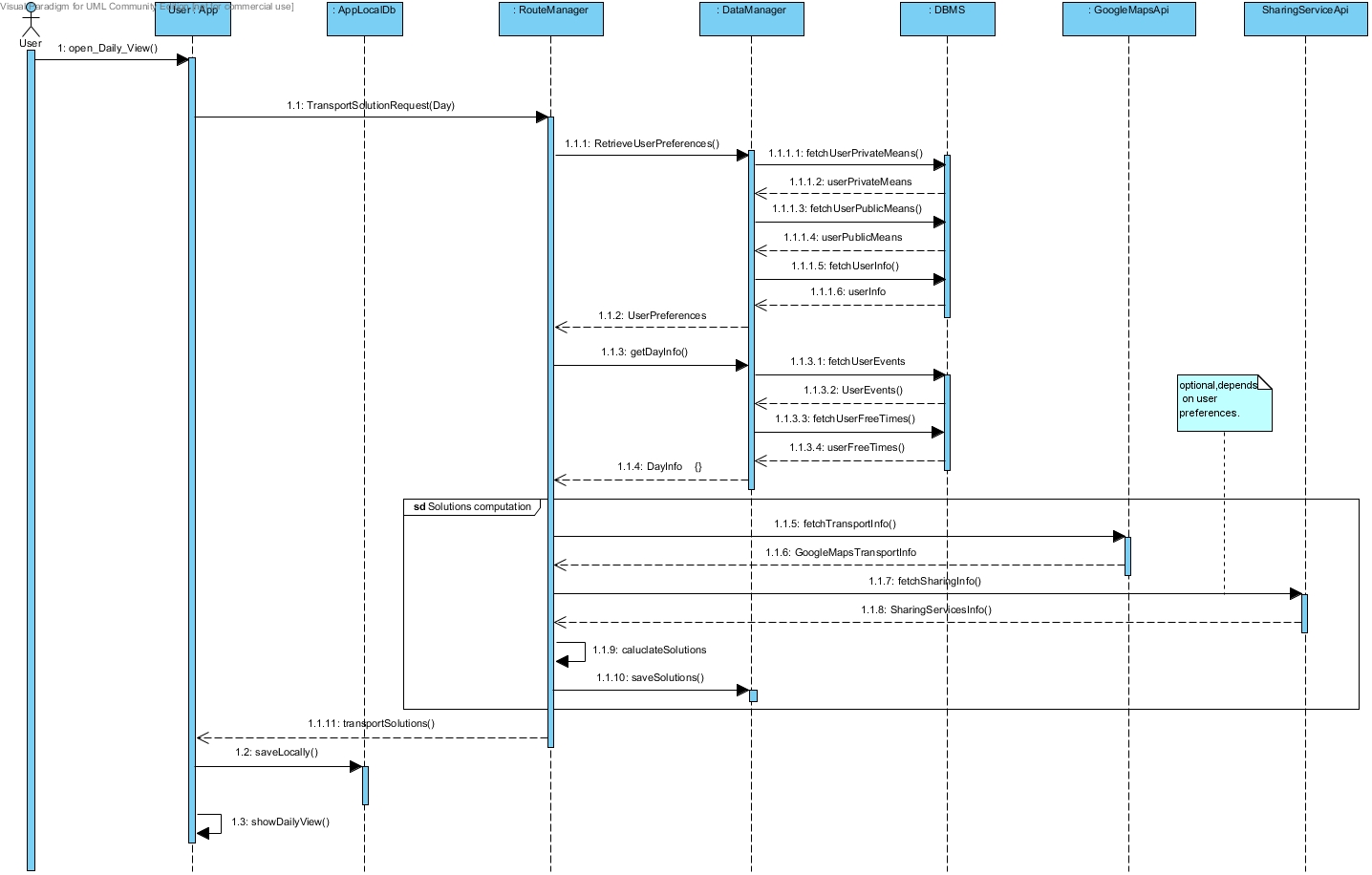
*Figure 6,*

## Runtime View

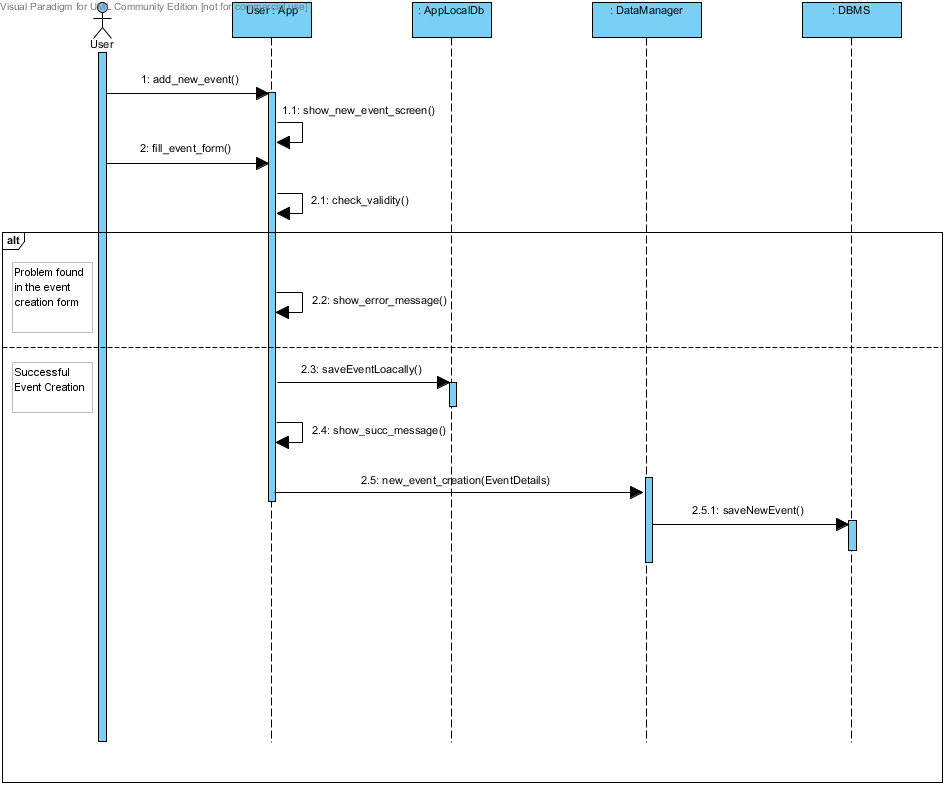
In this section we will describe the dynamic behavior of the system. In particular, it will be shown how the components previously defined interact one with another, using sequence diagrams. Beware that this is still a high-level description of the actual interactions that are going to take place, so functions and their names may be added or modified during the development process.



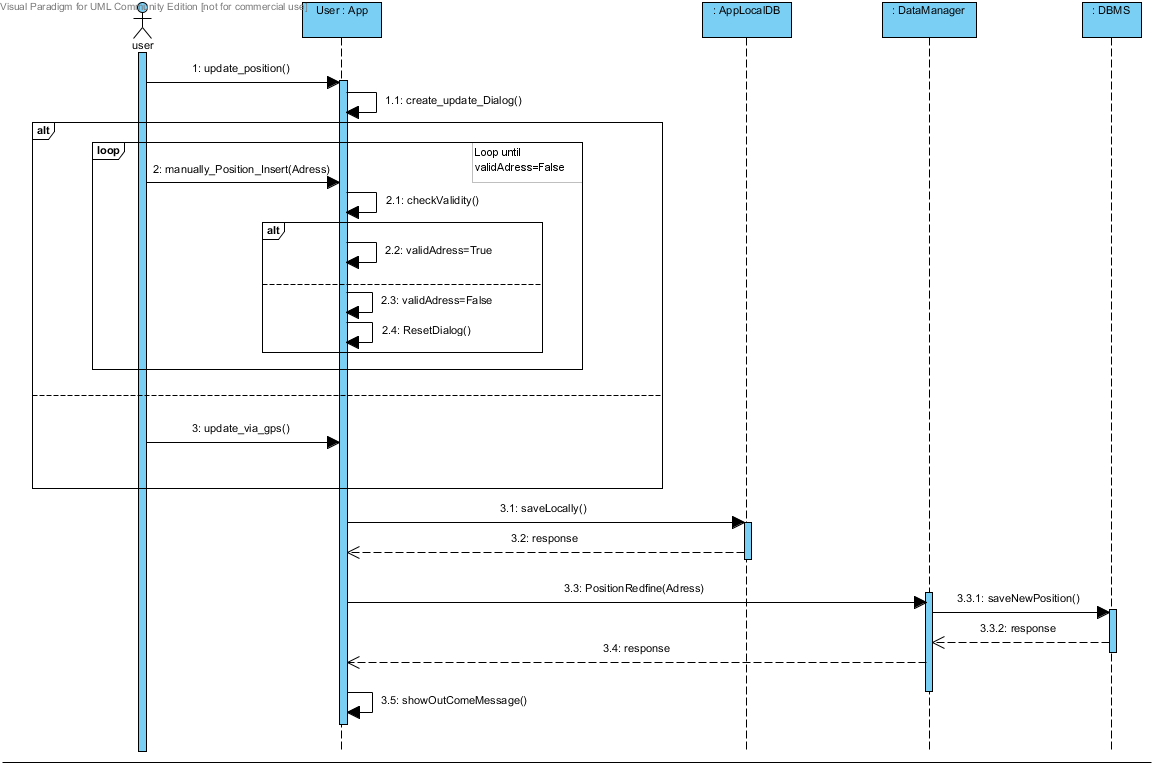
*Figure x, User Login*



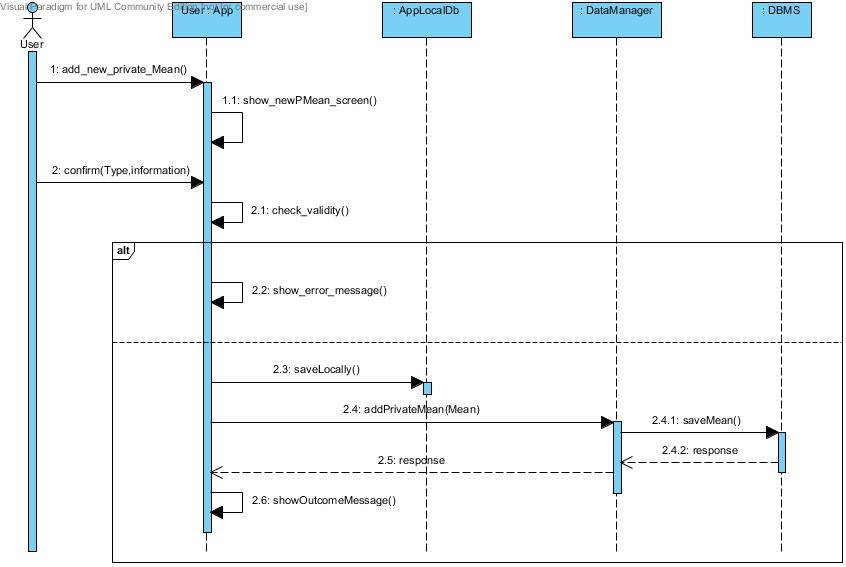
*Figure x, Transport Solution Calculation*



*Figure x, Event Creation*



*Figure x, Update Position*



*Figure x, Add new private mean*

### **Component Interfaces**

## Selected Architectural Styles and Patterns

#### **Architectural Patterns**

#### **MVC** (Model-View-Controller) pattern has been widely in our application. There are a lot of benefits on this choice, first of all the separation of the three components allows the re-use of the business logic across application and multiple user interfaces can be developed without concerning the codebase. Another crucial point is that MVC facilitates the developing process, for example The developers of UI can focus exclusively on the UI screens without bogged down with business logic and the developers of Model / business can focus exclusively on the business logic implementations, modifications without concerning the look and feel of UI.

Our Application server will use the SPRING framework, which is an MVC framework.

#### **Client Server Model**

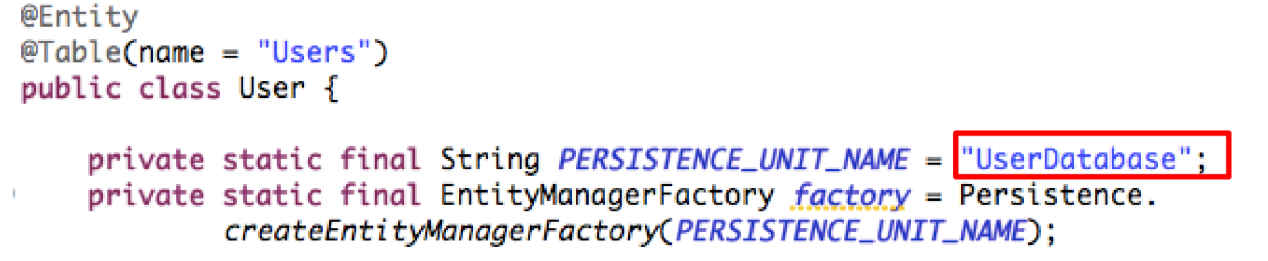
The application is strongly based on a Client-Server communication model. The clients being the mobile application in the first release. The clients are thin, thus to let the application run with low consumption of resources. All the computational processes relative at the transport solutions is powered server side. By the way the application is not only a UI layer, for example it has own local database, this is very important to maintain the persistence of information in limit conditions ( for example an user is able to insert an event in case of broken internet connection, then the mobile application will send the new information on server when it will possible). This approach is very important also because it improvers the maintainability of the system.

#### **Structural Design Patterns**

* *Façade Pattern* hides the complexities of the system and provides an interface to the client using which the client can access the system. (Used in some components of the component diagrams)
* *Bridge Pattern* decouples an abstraction from its implementation so that the two can vary independently. This pattern decouples implementation class and abstract class by providing a bridge structure between them. (Used in the interfaces of the component diagrams)

#### **Creational Design Patterns**

* *Factory Pattern* allows to create object without exposing the creation logic to the client and refer to newly created object using a common interface. It is particularly useful if applied in combination with the MVC pattern. We use this pattern in the implementation in several cases, for example with the JPA Entities like in the code snippet below.

**

#### **Behavioral design pattern**

* *Strategy Pattern* allows the changing of a class behavior or its algorithm at run time. This pattern, if applied correctly, guarantees a high level of polymorphism.

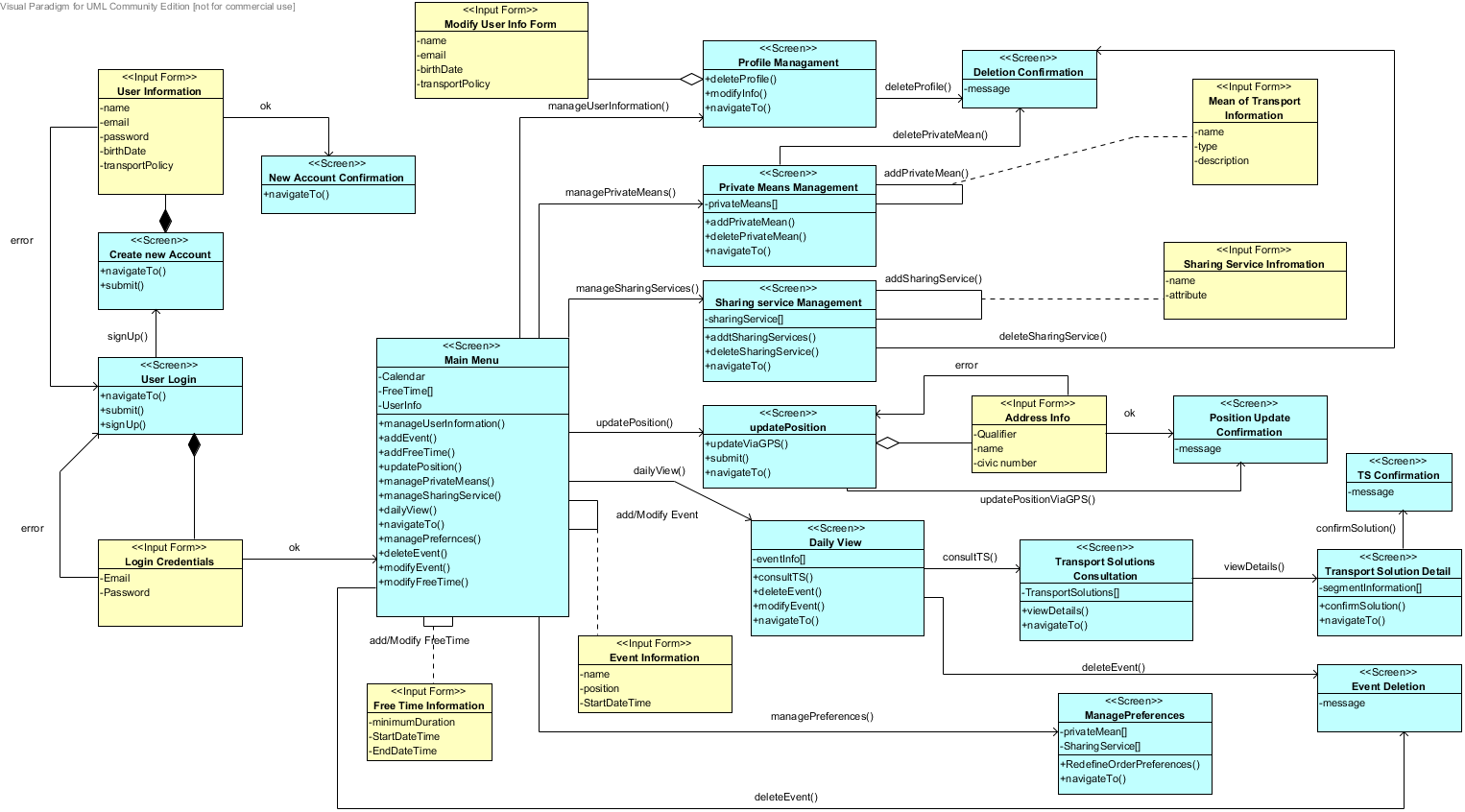
## Other Design Decision

# ALGORITHM DESIGN

# USER INTERFACE DESIGN

## Ux Diagram

The UX diagram in *Figure 1* shows the diﬀerent screens of the User Interface of the client Application and the interaction between them. In According to the RASD the user after logging in is able to manage is events, free time, preferences and means options in a very simple way. Obviously he can also view the Daily schedule, consult the transport solutions and update his position if he needs to do it.

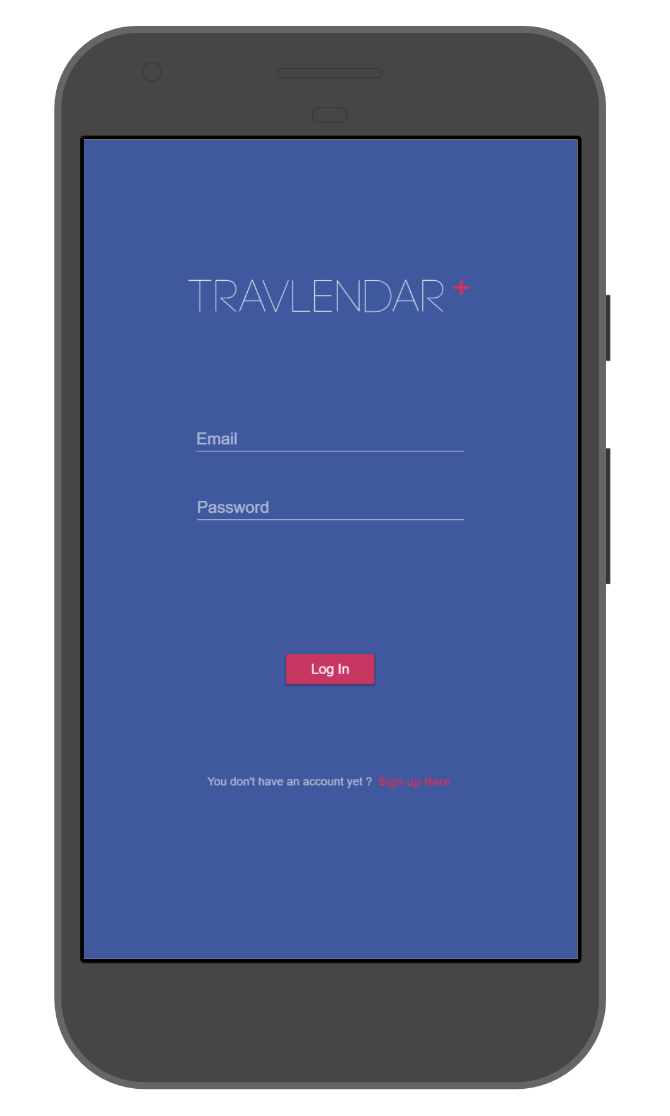
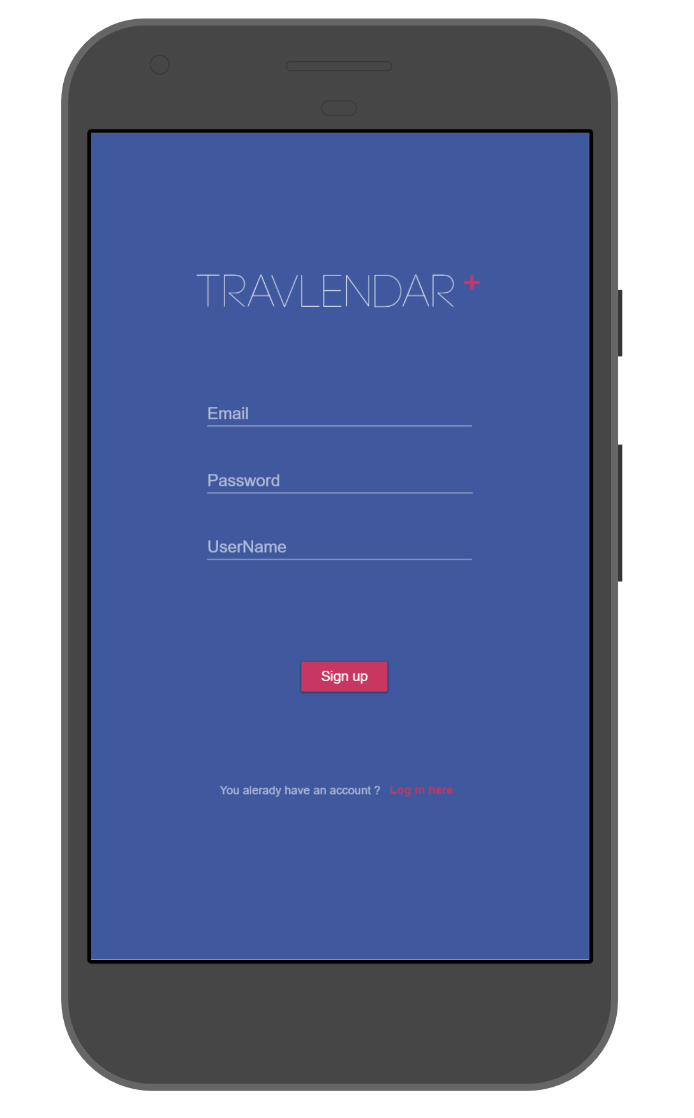


*Figure 1, Ux diagram*

## Mockups

In the RASD document we introduced a first prototype of mockups, after the definition of particular details and functionalities we propose in the following images the mockups of the Android Application.

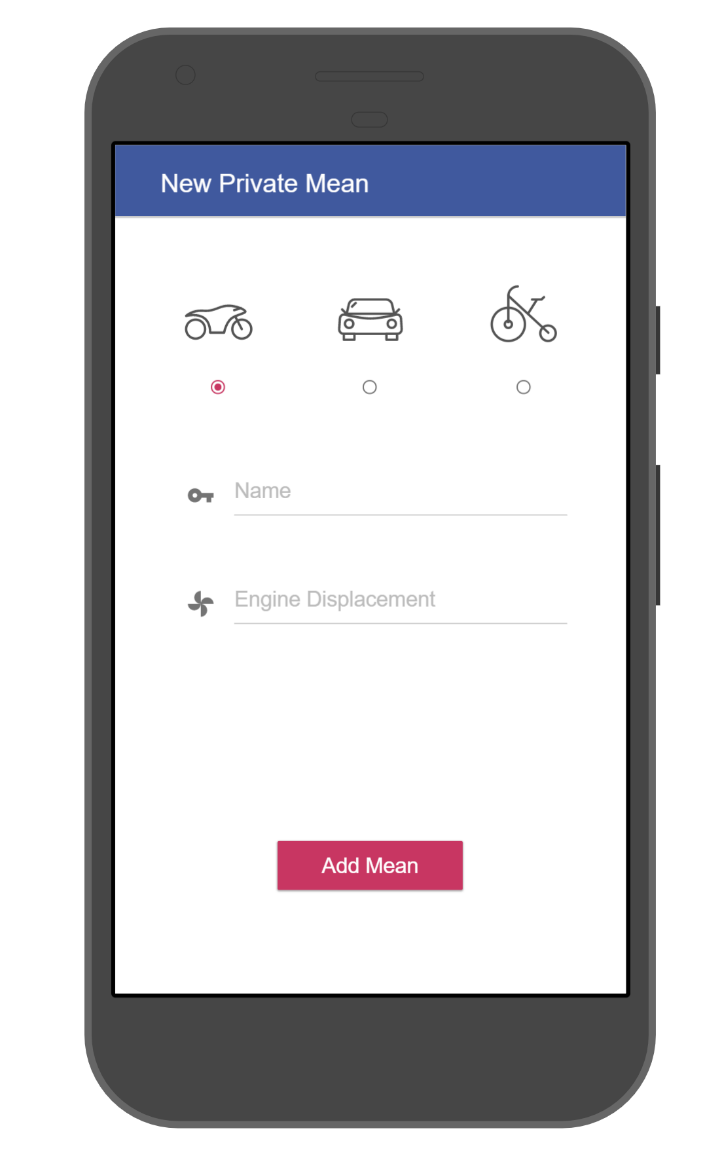
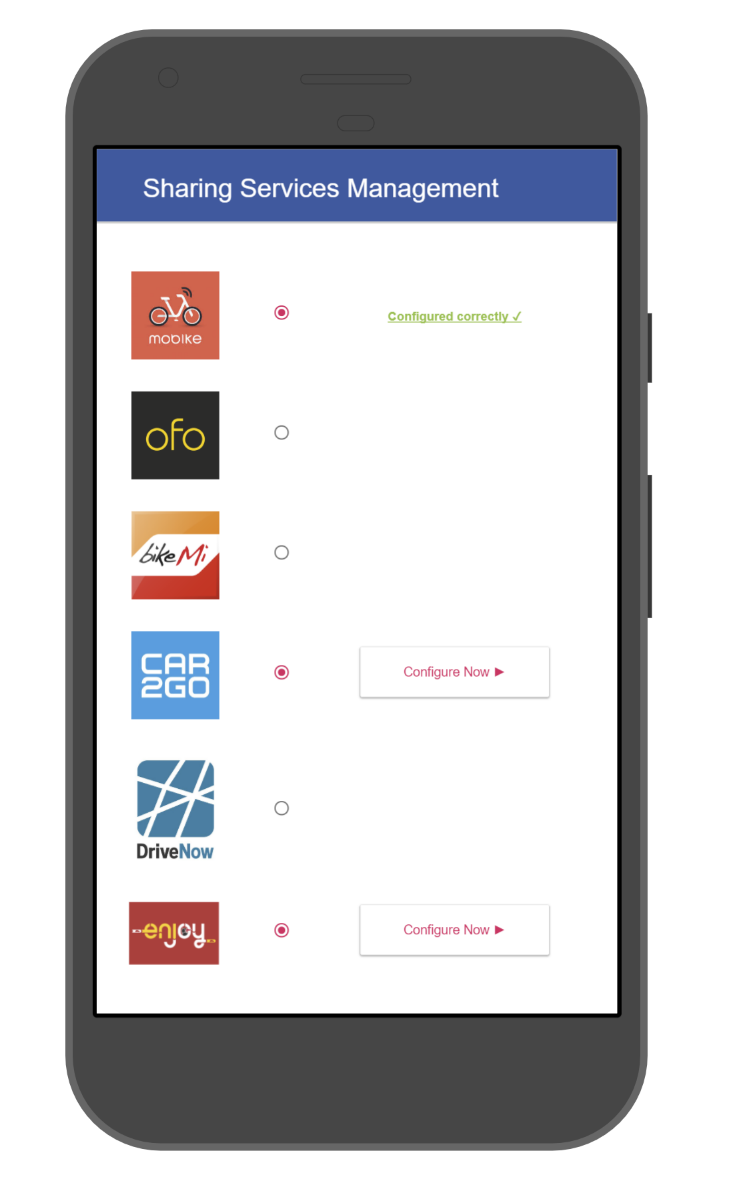
In *Figure 2* is shown the login form with the Email and password fields, it is also offered the possibility to switch in the sign-up activity (*Figure 3).*



*Figure 2, Travlendar Login*  *Figure 3, Travlendar Sign-Up*

In *Figure 4* is represented the screen responsible to adding private means of transport, the user must select the type of mean pressing the correct radio button, Insert the name and the engine displacement.

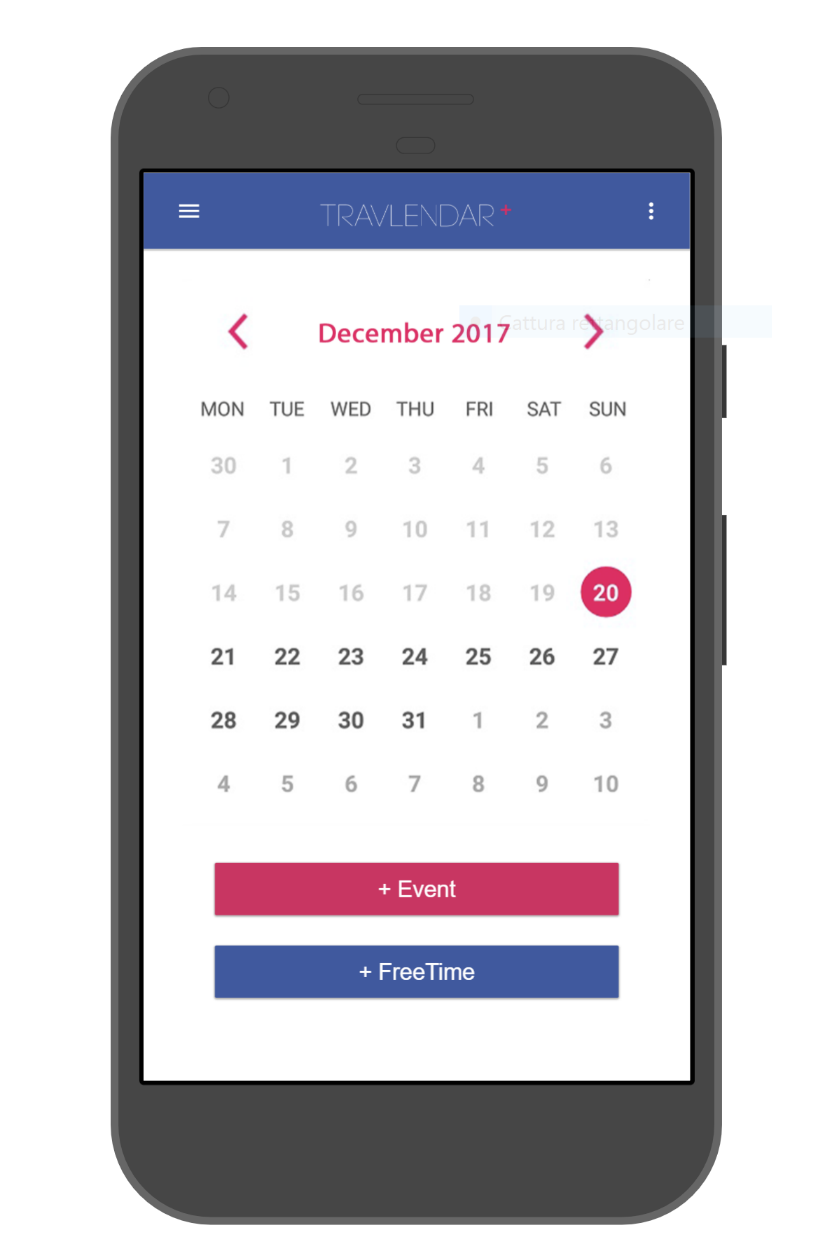
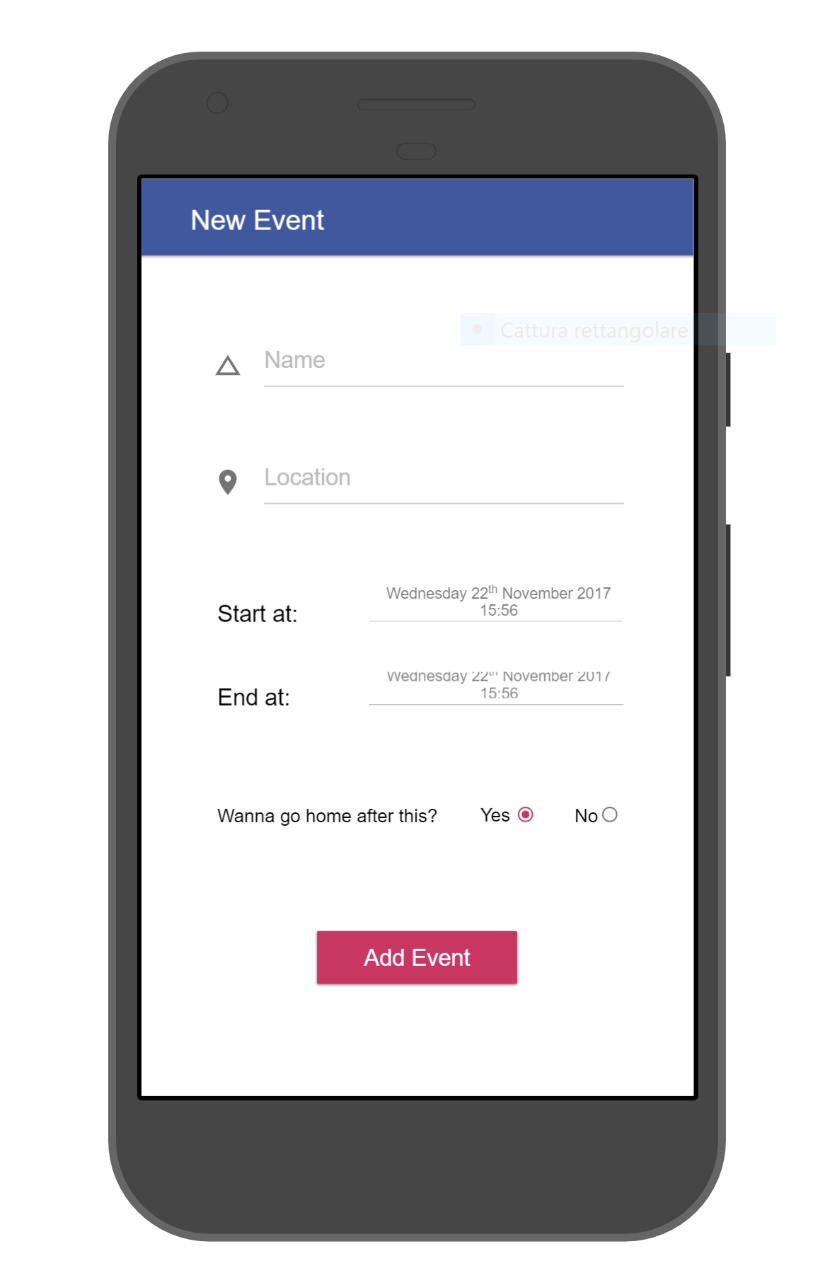
An user could manage the Sharing Services with the interface proposed in *Figure 5,* it will be populated with the available sharing services integrated with our systems.



*Figure 4, Adding private means*  *Figure 5, Managing Sharing Services*

The Screen in *Figure 6* represents the home activity of the application. With The button in the left side the user could open the personal menu (SideBar) with all the links to the other screens.

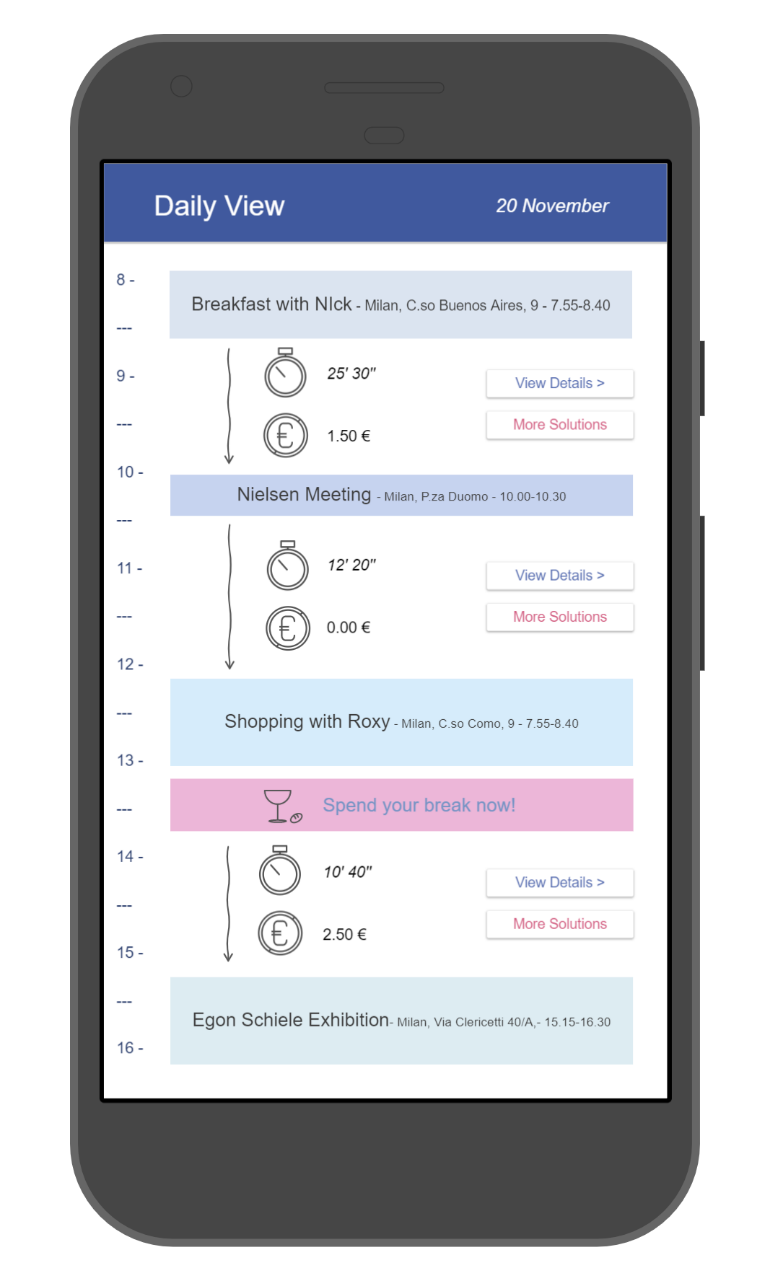
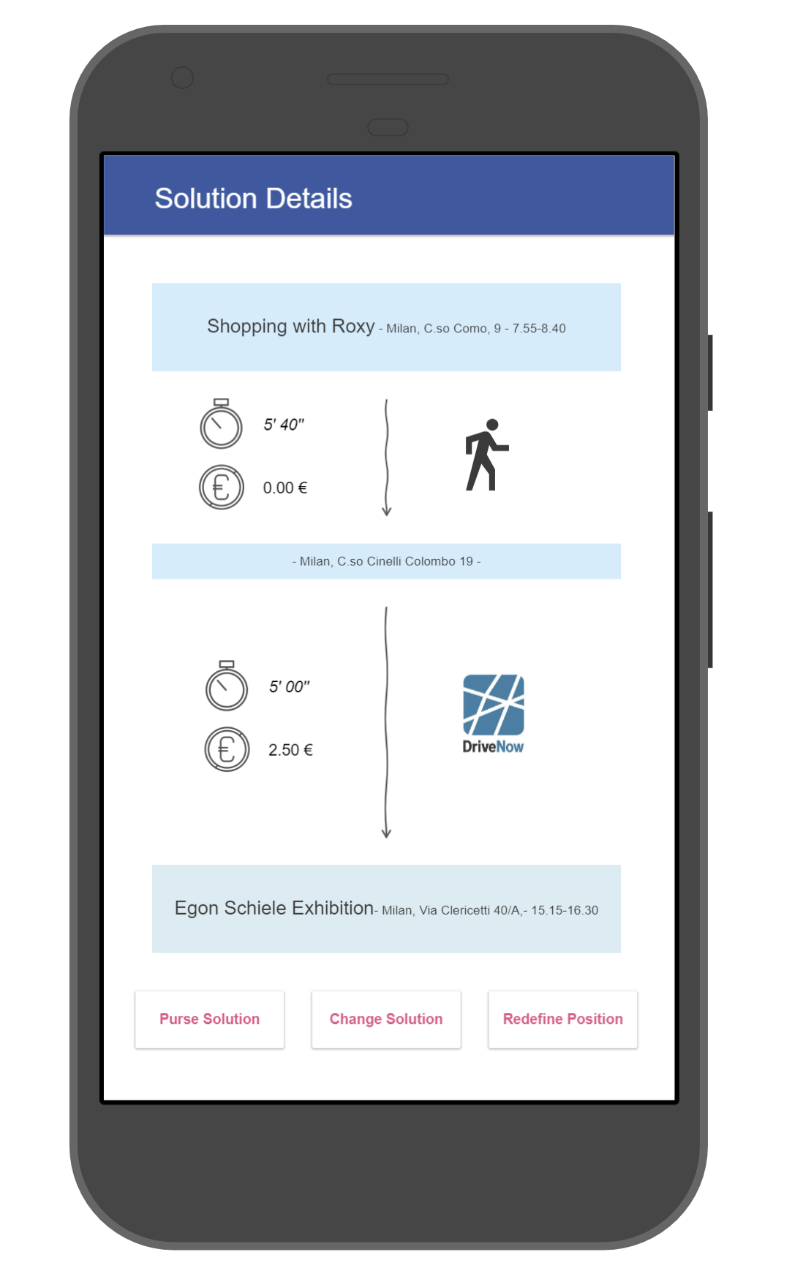
Pressing the Button “Add Event” the user is redirected in the screen represented in *Figure 7* where he is able to specifying a new event and add it in the calendar.



*Figure 6, Travlendar Home Figure 7, Adding a new event*

The Screen in *Figure 8* represents the Daily View where there are represented the event scheduled during the day and the best mobility option calculated by the system on the basis of the user preference.

The user is able to change solutions pressing in the “More solutions” button or view the solutions details and be redirected in the screen in *Figure 9*. Here he could analyze the details of all the segment that compose the solution. He could Purse the solution, change it or redefine his position if it changes from the last event.



*Figure 8, Daily View Figure 9, Solution Details*

In *Figure 10* is shown the transport preferences screen. The user is able to choose his favorite policy and choose the preference’s order of his private means and Sharing services previously added.

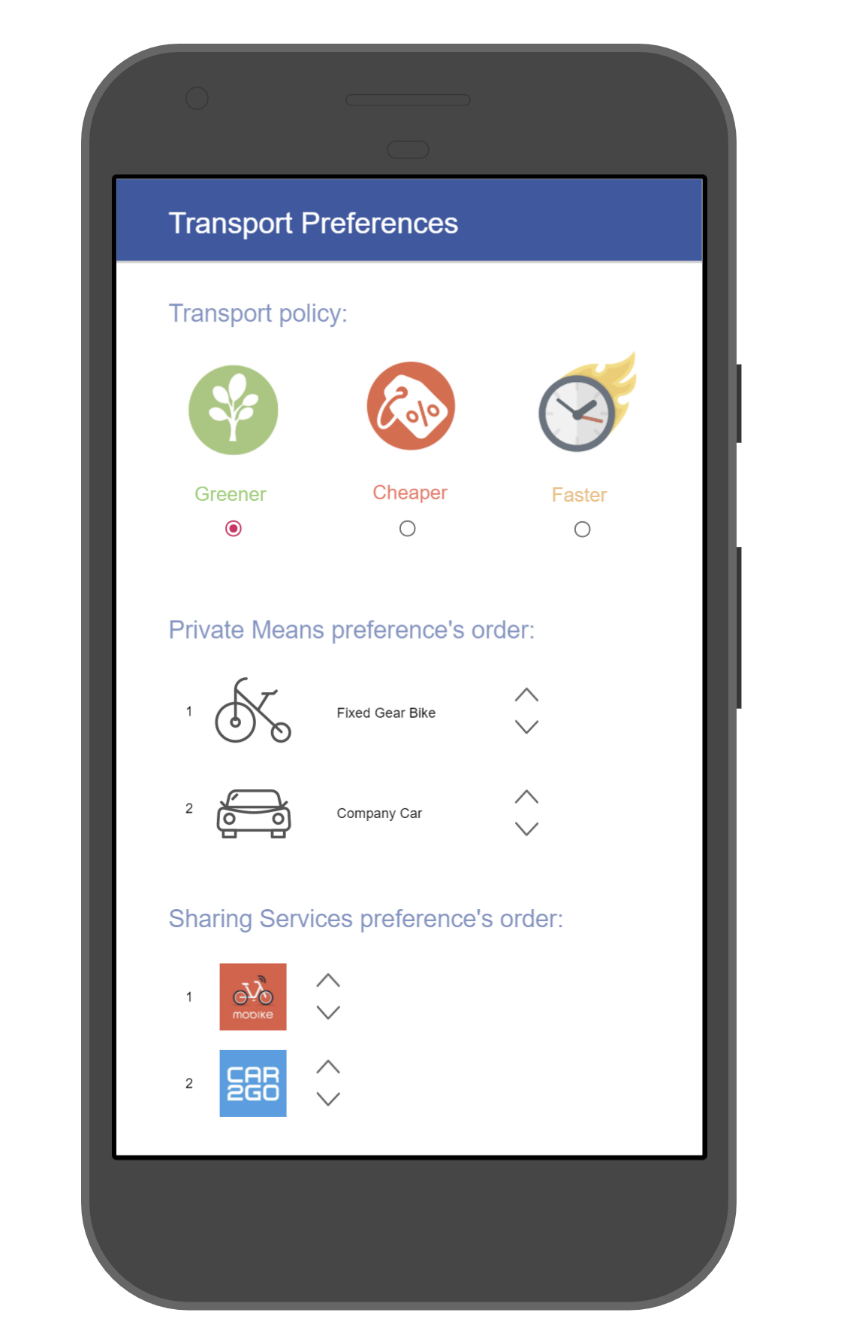


Figure 10, Transport Preferences

# Requirements Traceability

All the decisions in the DD have been taken following functional and nonfunctional requirements written in the RASD. The following list provides a mapping between goals and requirements defined in the RASD and system components illustrated in the DD.

**[G1]** Allow a visitor to became a registered user.

-Account Manager.

-Mobile App component.

**[G2]** Allow user to login to application.

-Account Manager.

-Mobile App component.

**[G3]** Allow user to create a new event in the calendar.

-Data Manager.

-Mobile App component.

**[G4]** Allow user to modify an existing event of his/her calendar.

-Data Manager.

-Mobile App component.

**[G5]** Allow user to delete an existing event of his/her calendar.

-Data Manager.

-Mobile App component.

**[G6]** Allow user to consult the transport solutions between events in the calendar proposed by the system.

-Data Manager

-Route Manager

-Mobile App component.

**[G7]** Allow user to re-define dynamically his instant position to re-plan the transport solution.

-Data Manager.

-Mobile App component.

**[G8]** Allow user to set free time (break) during the day schedule.

-Data Manager.

-Mobile App component.

**[G9]** Notifying events overlapping.

-Data Manager.

-Mobile App component.

**[G10]** Notifying the time-unreachable event.

-Data Manager.

-Mobile App component.

**[G11]** Allow user to configure transport preferences and external services to be used

-Data Manager.

-Mobile App component.

**[G12**] Allow user to set an event as ending event.

-Data Manager.

-Mobile App component.

**[G13]** Allow user to buy a ticket or to reserve a mean of transport of a suggested solution.

-Data Manager.

-Mobile App component.

# IMPLEMENTATION, INTEGRATION AND TEST PLAN

# EFFORT SPENT

Agostini Andrea:

Ciampiconi Lorenzo:

Es-skidri Rachid:

# REFERENCES

[1] Google, Android Developers - Design https://developer.android. com/design/index.html

[2] Software Engineering 2 Project, AA 2017/2018 - Project goal, schedule and rules

[3] Software Engineering 2 Project, AA 2017/2018 - Assignments 1 and 2

[6] Software Abstractions (Logic, Language and Analysis) – Daniel Jackson

[7] Il linguaggio Alloy nella specifica formale di modelli UML – Tiziano Verone