POLITECNICO
MILANO 1863

# Travlendar+
## Design Document

**Bolshakova Liubov**
**Campagnoli Chiara**
**Lagni Luca**

**Software Engineering 2**
**November 25, 2017**

# Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to provide information about the design decisions we made for the development of the system Travlendar+.
It is intended to be a more software and system developer-side detailed document of what was proposed in the RASD.
This document includes:

- An overview of the architectural design.
- Describtion of the main components and interactions among them.
- Basic design patterns to be included in the final implementation.
- Juicy algoriyhms on which the software relies.
- Overview of the user interfaces.
- Integration test plan.
- Implementation plan.

## 1.2 Scope

Travlendar+ is a calendar based applications, which allows the user to create meetings with different locations and computes the best way to reach such locations.
It includes different means of tansport, public, private or personal, autonomous or not.
Among its functionalities there are the possibility of creating flexible breaks between meetings, deny travel means the user does not want to use and choosing the combination of travel means which minimizes the carbon footprint (if required).
Besides, it supports the user in the reservation of cars and bikes of a vehicle-sharing service and in the purchase of tickets of public transport companies.
More details about the goals and functionalities of the system can be found in the RASD.

## 1.3 Definitions, Acronyms, Abbreviations

- API: Application Programming Interface.
- DD: Design Document
- GPS: Global Positioning System.
- GSM: Global System for Mobile Communications.
- GUI: Graphical User Interface.
- OAMOT: Other Autonomous Means of Transport
- ONAMOT: Other Non-Autonomous Means of Transport
- OS: Operating System.
- RAM: Random-access memory.
- RASD: Requirement analysis and Specification Document.
- SMS: Short Message Service.

## 1.4 Reference Documents

- Mandatory Project Assignments.pdf
- Requirements Analysis and Specification Document
- Design Deliverable Sample from A.Y. 2015-2016.pdf
- DD From the car sharing project.pdf
- Integration and test plan from the car sharing project.pdf

## 1.5 Document Structure

After the introduction, this document is divided into five main parts:

1. ***Architectural design***: it contains all the main decisions about the general architecture of the system. In particular, here is specified the tier division of the proposed system and are included the main diagrams (component diagrams, deployment diagrams, runtime diagrams and the complete class diagram). Here are also described some of the possible design patterns which can be used for implementation.
2. ***Algorithm design:*** it includes a description of some of the main algorithms that will be used to implement our system.
3. ***User interface design:*** it inludes some mockups of the mobile application interface, undelying how the main functionalities of the system can be accessed by the user.
4. ***Requirements traceability:*** it explains how the requirements identified in the RASD have been fullfilled in the design elements and decisions.
5. ***Implementation, integration and test plan:*** it explains the order in which we are going to implement, integrate and test the components of the system.

# 2 Architectural design

## 2.1 Tiers

Travlendar+ is based on three main tiers (which will be expanded later).
A diagram of the proposed system was already present in section 2.4.3 of the RASD; here we provide on the same diagram a division between the different tiers and a more detailed description of each one of these.

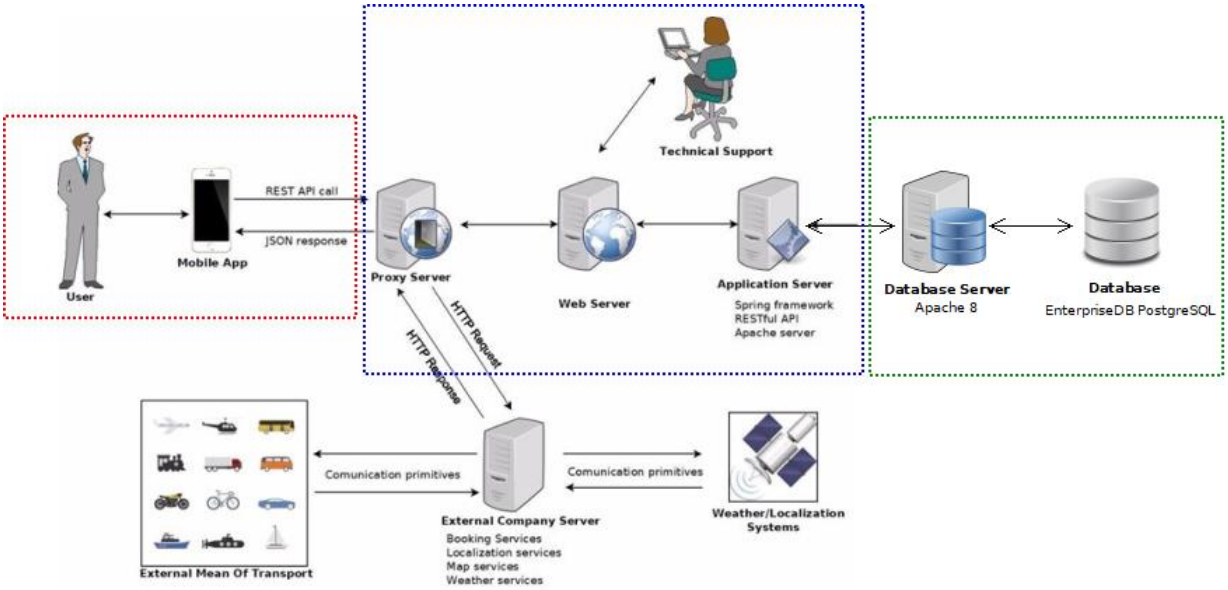


Figure 1: Proposed system with tier division. Red: frontend, Blue: backend, Green: data

The ***fontend tier*** consists of the mobile device on which the application runs.
The ***backend tier*** is mainly represented by the application server, where main decisions and computations take place, although a small part of logic is left to the mobile application (simple elaborations of data such as the location of the user through GPS, or remodeling of the view presented to the user).
Also firewalls and proxyies (as well as load balancers or others useful systems) are part of this tier. Finally the ***data tier*** handle the operation that concerning data store and retrivement, providing the appropriate hardware for doing that.
This is a very simple distinction, soon we will se a more detailed architecture that we have proposed for the system.

## 2.2 Layers

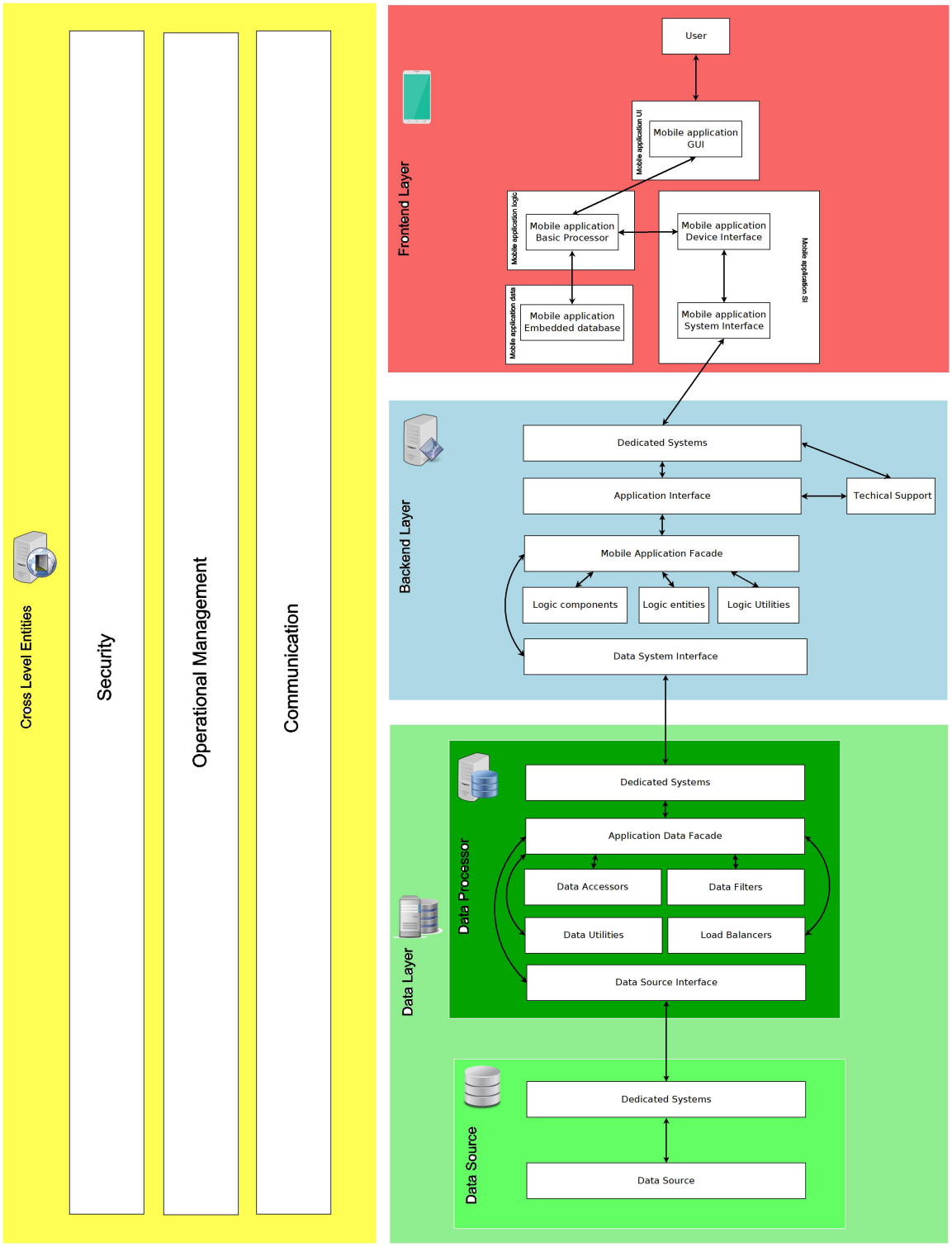


Figure 2: Layers diagram

We will do only a fast overview of this diagram because we think that it is enought self explanatory.
The ***frontend layer*** represent the classical idea of client tier, including a GUI and a lightweight processing power that allows the user to interect with the rest of the system and have a more fluid experience.
This fluidity is done by delegating some operations directly to the client side application in order to not overload the server with useless operations.
The ***backend layer*** is concercning the application running on a server as well as all the utilities that this one uses to perform its tasks.
And, finally the ***data layer*** concerns the complete management of the data used by the application, from it's manipulation to it's storing or retriving.
It's important to specify that, in the final implementation, we will have two different data layers, one delegated to a database system which will be specifically set up for this and one that will be putted in the mobile device of the user in order to keep some important informations (especially if the user intends to se them more than one time) for avoiding main databases overloads.

# 3 Components

## 3.1 High level component view

We can see that , from a very high point of view, the application seems to be pretty trivial.
There are three main components:

1. User
2. Server
3. Database

The User interact with the application engine throught a mobile device in which is installed our mobile application.
Also the mobile version of our application has a small functional power and limited database capabilities, as we have said before.
The mobile application delegate complex tasks to the main application engine which relies on servers setted up for this tasks.
The server application needs also to integrate user data with data the this one has previously saved or that it needs that comes from external sources.
This external datas are provided by external companyes that interact with our application (or, better, we interact with them for the data) via API.

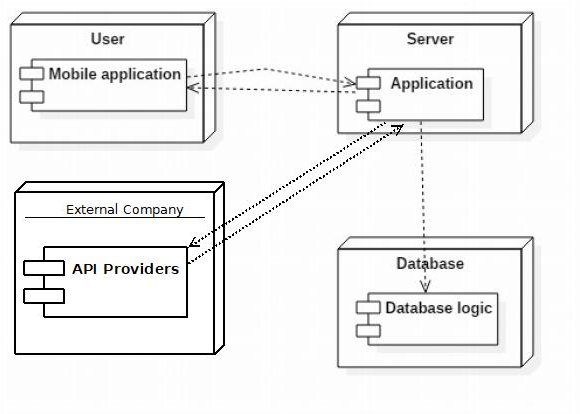


Figure 3: High level component diagram

## 3.2 Detailed Component view

We can see that our external actors are users and external companies.
The user , registred (client) or not (visitor), interact with our application in the client side of our server-side application.
Then the application interract with the server.
On the server side , the connection handler manage the requests that the users do, enshuring that , for each user, only his data will be processed and that other users cannot see that.
So, once the connection is established and the "user-environment" is setted, the UserController decide which operation to perdorform according to the command that the user has send to the server.
At this level, there are several possible operations that can be done and , in the diagram, it's possible to see which controller will manage them.
The user device controller is the one which manage what concerning the usage of user's device data like coordinates , messagges and so on.
The meeting controller is the one which duty is to manage the meetings of the users and it interacts strictly with the pathcontroller which is the one whose job is to allow the userto choose a path and a mean of transport to use along that path.
It has also the duty to inform the user in case of problems during the path or in case of problems concerning the mean of transport.
For doing its job it needs to interact with even two more controllers: the NavigatorController which is the one that , according to the user preferences , choose a list of possible MOT that can be used in that path, sorted so the first choices will be the ones that are more suitable for the user preferences, and the BreakConstroller who is the controller that manage the breaks (and the alarm if necessary) during a path.
The UserPreferencesController is the controller used for performing operations concerning settings of the user.
Finally we have the LoginController whose job is pretty trivial to imagine.
Then , as we have said before, there is another kind of actor for our application: the external companies.
We interact with them via API and we need to do so for retrivening infos about the means of transport, rides, tickets, strike notifications and so on.
Our system has disposed a different way to interact with them via a dedicated channels.
Now , reather than a connection handler, we have an APIManager which is the one that interface with the API provided by the external companies and extract infos.
This infos are then passed to the ExternalMeanOfTransportCompanyController (in case that this data is related to a mean of transport provided by a company) or ExternalCompanyController (in case that this data is not related to a mean of transport company).
The ExternalMeanOfTransportCompanyController interact with NotificationController, MeanOfTransportCon-

troller and RideController for sending notification, manage a mean of transport and dispose some rides for the provided means of transport rispectively.
The ExternalCompanyController interact with the NotificationController (for the same reason we have explained before) and the PathRestrictionController for setting restrictions to some paths.
There i also the SystemSharedController which is the one that manage obects that are shared in the applicaton.
All this controllers must be able also to manager software anomalies that can be encurred during the usage of the application and , all of the , uses the needed object building them brand new (when needed) or retriveing them from external databases if they're not available yet.
At this level we have avoided to show the design pattern classes in order to make the diagram more redable without loosing the functionality of the application.

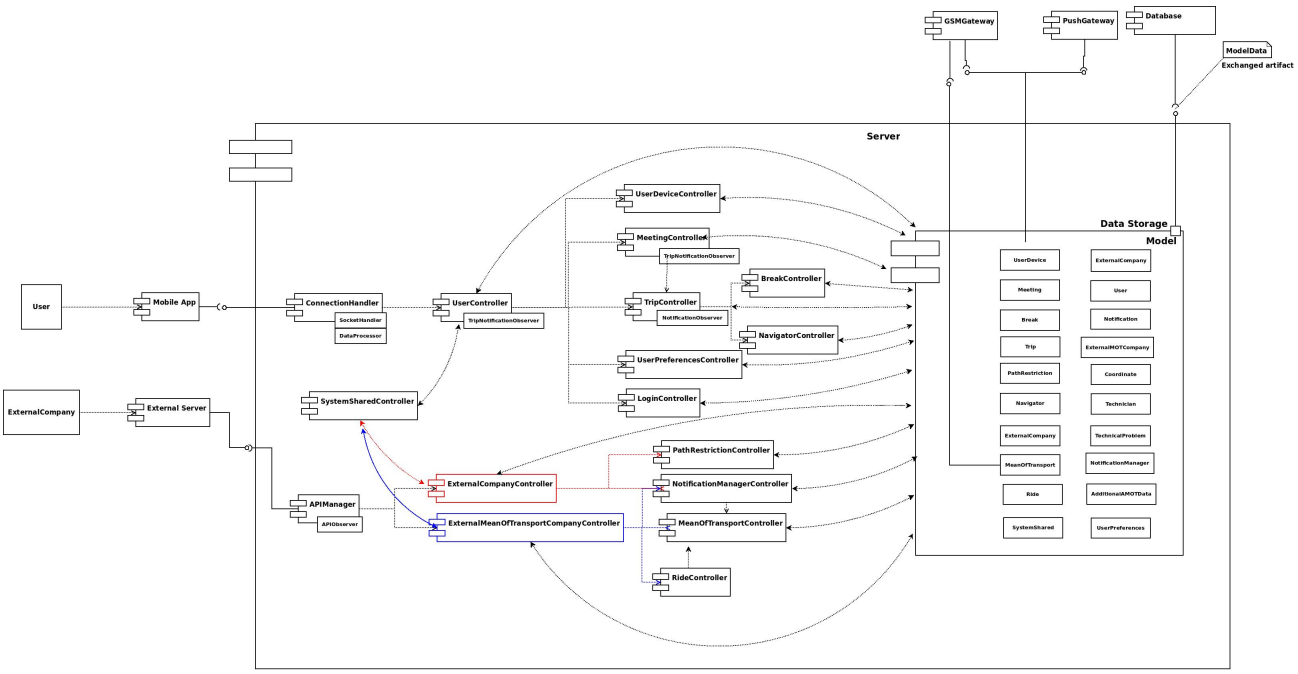


Figure 4: Detailed component diagram

## 3.3 Deployment View

As we have said before, there are more than only three tiers under the surface.
In this section we will explain in more details why we have decided to use these solutions.
For accessing our application via a web browser we have disposed an interface using AngularJS and Bootstrap for a desktop-like application interface.
In this part it's not possible to use completely the application, only a subset of functionality.
But the application itself needs a mobile device to run.
We have used Titanum Appcelerator with Bootstrap Framework to make the client side application , with some limited functionalities.
The embedded database has been done using mongodb with a subset of the tables used in the main database in order to make faster.
There are some firewall (Cisco Catalyst) placed in some hot points in order to prevent some attacks or bad actions that can compromise the functionality of our application or can compromise the privacy of our users.
Then we have some proxy servers, they're used to select which requests can be processed (and to which part of the system delegate them) and wich one should be discarded.
This is also a useful behaviour to prevent attacs that have not been blocked previously by firewalls.
There are also some load balancers (jetNexus) that are usefull for managing large amount of client requests.
They're a very important part of our application because this is intended to be a very large scale application , with a lot of user (and requests) and the speed of the responses should be maximised as much as possible in order to provide an almost real time application and, a way to do so , is to divide requests and task omogenously among our servers that are disposed for performing that specific task.
Then we have disposed different servers for technical support and for external companies.
These servers for external companies are not only external companies server but are also some of our servers that have been disposed for chacing external informations for a faster retrival of them. Our applications servers uses spring frameworks and use Apache Tomcat as application server engine.
On this machines there is the core of our application and in here the majority of the complex tasks are performed.
And, finally we have disposed some RDBMS servers and systems that uses the EnterpriseDB technology,we have decided to use this RDBMS because it provides a facilitate support for string JSon data (as well as MongoDB does) and also for normal relational informations.

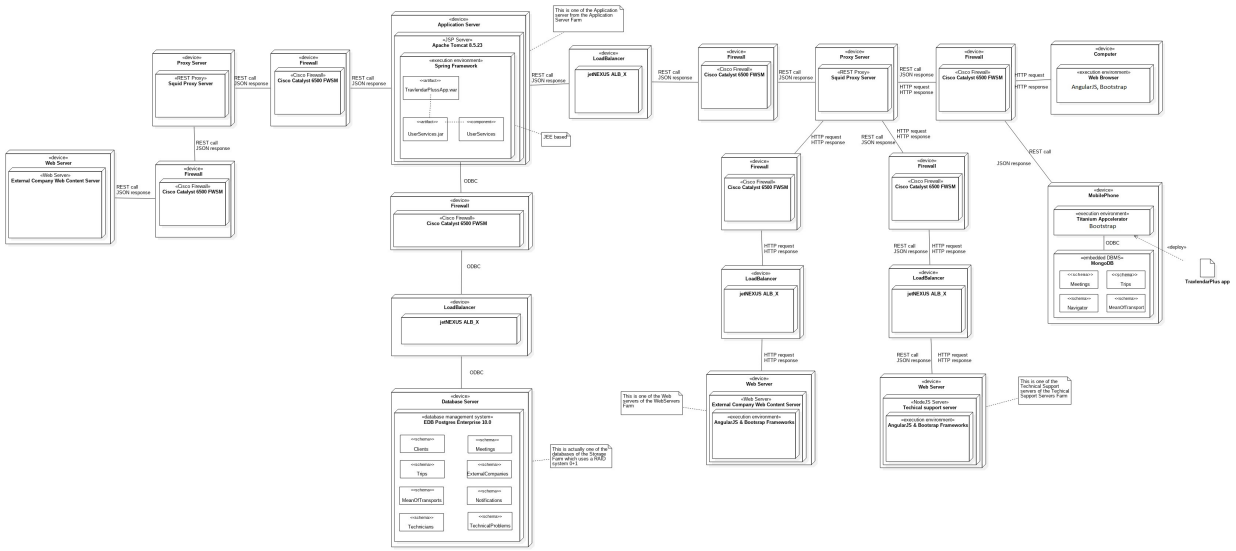


Figure 5: Deployment diagram

## 3.4 Runtime and Activity view

### 3.4.1 Meeting creation

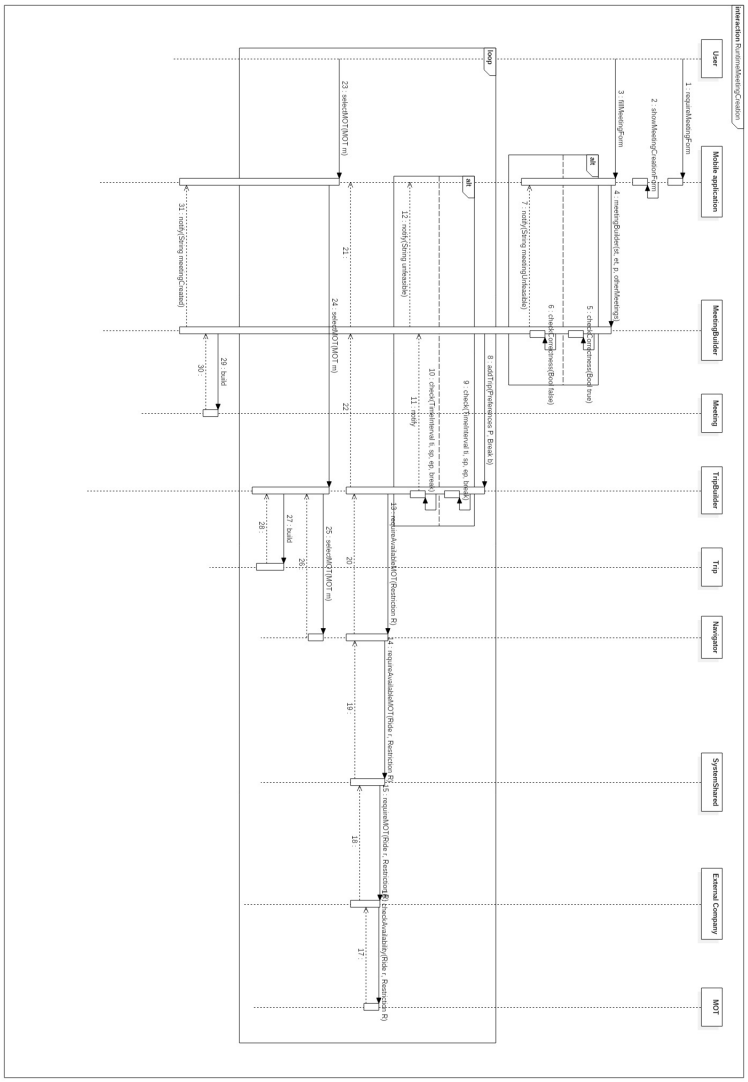


Figure 6: Runtime sequence diagram of meeting creation

This sequence diagram describes the runtime interaction between the components during the creation of a meeting. It refers to use cases 3, 4, 5, 17, 18, 19 and 21 described in section 3.2.2 of the RASD.
We illustrated the more general case in which the user is a visitor, that means an unregistered user: this means that he or she is asked to insert his preferences and constraints (for example deselect unwanted travel means or choice of the most ecological route) as inputs for the meeting creation. The only difference with a client, or registered user, is that these settings may have been saved, if the user has chosen to, and they will be searched for in the database.

### 3.4.2 Vehicle sharing reservation and ticket purchase

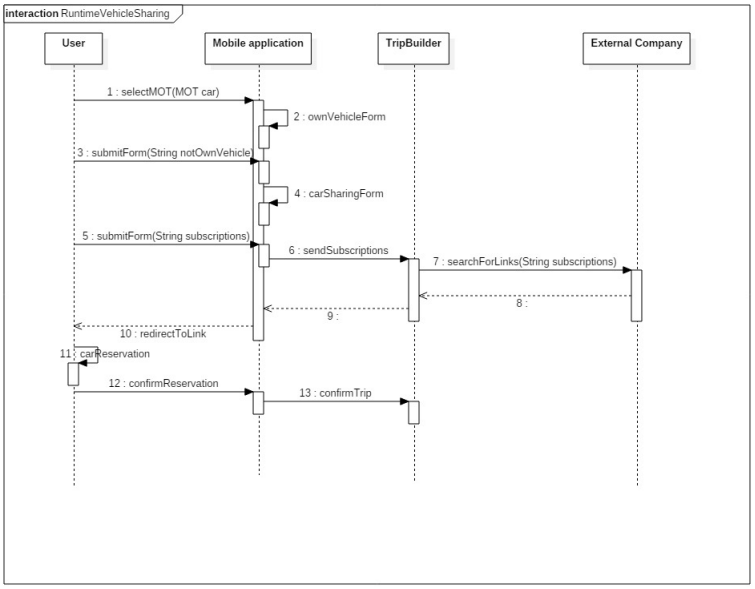


Figure 7: Runtime sequence diagram of shared-vehicle reservation

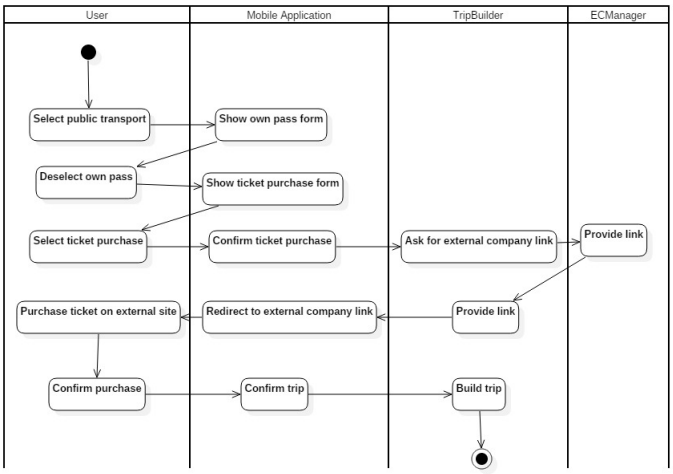


Figure 8: Runtime activity diagram of ticket purchase

The top sequence diagram describes the runtime interaction between components during the reservation of a vehicle of a sharing service. A vehicle can be a car, a bike, a motorcycle or anything provided by external companies interacting with the system. It refers to use cases 13 and 14 of section 3.2.2 of the RASD.
We are in the situation of a meeting creation and the user is selecting the chosen travel means: one of these is a vehicle and the user has not a personal one. The reservation of a vehicle is perfomed on the external company's website or application; the user is asked for the list of companies for which he or she has a subscription. Once the reservation is complete, the user confirms the creation of the trip. A very similar interaction is the purchase of a ticket for a public mean of transport (use case 7 of section 3.2.2 of the RASD), shown in the bottom activity diagram: the user is selecting the travel means during the creation

of a meeting and one of these is a public mean of transport. The user has no personal pass for such transport company. The user is redirected to the company's link, where the purchase is performed.