Integrated Mobility Management System

Nuno Miguel Silvestre Cameira
nuno.cameira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2018

Abstract

Several of transportation options exist nowadays when someone wants to travel from one place to another, but there is no way for someone to search specific options only available within that specific organization where one studies or works, while at the same time searching for general transportation options (e.g. public transports). To make this possible, we devised a configurable and extendable mobility portal, to be deployed within an organization, which enables its users to find the best transportation options to/from/between several sites, most often different facilities of the organization. We implemented a system that provides this functionality by querying external services that provide information on routes and respective schedule, with visual representation of the results and map interaction. We also implemented our own Carpooling service and respective plugin that is connected to our Mobility portal. An evaluation showed people could use and understand the whole system, thus certifying we had achieved our initial objectives.

Keywords: carpooling, mobility portal, trips, routes, driver, passenger

1. Introduction

Nowadays there are many transportation options available when someone wants to travel from one place to another. With all these possibilities it becomes difficult to search for the most convenient option (either in financial terms, travel time or even the distance covered) for a particular situation. People can search for generic transportation options in several systems, the most well-known being Google Maps, but what if we look, for example, at the college students, how can they search for transportation options that only their college provides and only they can use? Or in companies with several facilities where employees must commute from their homes to a given facility or between facilities? There is not a particular system that allows this specific transportation options while at the same time searching for general transportation options (e.g. public transports), thus we propose to develop one.

In this work we developed a system able to query different services for routes and their schedules in order to find the one that best suits the user's needs. This system which we called Mobility Portal, should be able to be deployed on any organization that provides their own transportation options by adding custom plugins that can query those specific transportation services. This allows a more customized service for their employees.

Since this system aims to be deployed on any organization and assuming the system is already deployed on a specific one, this means that the users that will use this system are all part of the same organization. This is a very interesting situation and has a lot of potential, because not only it will be easier for users to trust each other, but also, because most of the times they share the same starting or ending point, which is the location of the organization where they both belong to, making it much more effective. Therefore we thought that it would be interesting to develop a Carpooling system and use it on our Mobility Portal. In order to do this we developed a plugin for the Mobility Portal that can communicate with this Carpooling system.

Carpooling is basically the sharing of car journeys so that more than one person travels in a car. It's inexpensive, since in carpooling usually the travel expenses are divided between all the occupants of the vehicle, it's environmentally friendly and contributes to the reduction of traffic congestion since fewer vehicles will be needed, which means less emissions per passenger thus less pollution. There is also the case of a social benefit for both driver and passengers. With all these advantages why should anyone drive alone? With this question in mind we want to encourage users to do carpooling. For this Carpooling service we analyzed different existing solutions and compared them in order to take advantage of the best advantages and avoid their drawbacks.

Thus our two main objectives are:

- Create a modular system (Mobility Portal) that allows users to define an origin, destination and departure date and search for different ways to travel from one point to the other on that departure date
- Build our own Carpooling service and add the respective plugin to our Mobility Portal that can query that service

A requirement underlying all our developments is that both the Mobility Portal and the Carpooling system should be able to be deployed on any organization with minimal effort.

2. Related Work

In this section we reviewed background information including existing approaches to carpooling and mobility portals that related to our work and the different technologies that could be used to implement our systems.

All the mobility portals we analyzed (Arena Mobility portal[1] and 511 SF Bay[2]) presented the same interface logic where the user can search for routes from a place to another, with a departure date. The results of this query are a set of routes that are displayed on a list and can be viewed on a map for better visual understanding. The 511 SF Bay also had one interesting feature which was on top of the map (were the results are displayed) there is a yellow horizontal bar where reports of accidents are shown, this same accidents are also displayed on the map with a yellow sign. This particular information on accidents is very helpful for a user to choose between different options. Although the Arena Mobility portal had one major drawback that we did not want to have in our Mobility Portal, which is locking one point, either the origin or destination point needs to be the Amsterdam Arena.

After analyzing several carpooling systems (Boleia.net[3], Blablacar[4], Poparide[5], SINGU[6]) we concluded that they all share the same basic interface to search for trips, which is a starting and ending point plus an optional departure date. Some allowed the login with other system (e.g. Facebook) which we also implemented on our Carpooling although not with Facebook but with Fénix.

Poparide had an interesting and appealing interface where a map with points representing trips that begin or end in those points is displayed (the bigger the point size the more trip options available). This allows the users to have an idea of what trips are available, even without having done any search.

The SINGU system had one major drawback which is the destination of a trip is locked to three possibilities (Tecnológico e Nuclear (CTN), Taguspark and Alameda) which decreases flexibility. What we said for our Mobility Portal also applies for our Carpooling system, thus it also will not lock the choices of either origin or destination. However this system presents some positive aspects like the ”Add waypoint” functionality which adds a location between the origin and the destination, thus allowing it to be found if a user searches that waypoint as destination

We took the basic interface as a starting point for our Carpooling system and added several relevant aspects of these systems into it: upload a picture of their car, add a recurring trip as a driver, define spoken languages on the user profile as well as preferences (e.g. likes to talk, does not transport animals)

We analyzed several different technologies that could be used to implement our systems.

The Representational State Transfer (REST) architectural style provides important qualities like: performance, scalability, simplicity, modifiability, visibility, portability, and reliability[7]. Nowadays almost every project or application provides a REST API, such as Twitter, YouTube or Facebook. It is the most logical, efficient and widespread standard in the creation of APIs for Internet services. It relies on a stateless, client-server, cacheable communication protocol and in most of the cases, the HTTP protocol.

The Spring Framework is an open source application framework and inversion of control container for the Java platform. The Spring Web model-view-controller (MVC) [8] framework is designed around a DispatcherServlet that dispatches requests to handlers. These handlers are based on the @Controller and @RequestMapping annotations, offering a wide range of flexible handling methods. With the introduction of Spring 3.0, the @Controller mechanism also allows the creation of RESTful Web sites and applications, through the @PathVariable annotation and other features.

Like Struts[9], Spring MVC is a request-based framework. The framework defines strategy interfaces for all of the responsibilities that must be handled by a modern request-based framework. The goal of each interface is to be simple and clear so that it is easy for Spring MVC users to write their own implementations, if they choose to do so. MVC paves the way for cleaner front end code.

Thymeleaf[10] is an modern open-source Java XML/XHTML/HTML5 template engine that can

work both in web (Servlet-based) and non-web environments. It is better suited for serving XHTML/HTML5 at the view layer of MVC-based web applications, but it can process any XML file even in offline environments. It provides full Spring Framework integration.

In web applications Thymeleaf aims to be a complete substitute for JSP, and implements the concept of elegant Natural Templates: HTML that can be directly displayed in browsers and that still work correctly as static prototypes.

The Google Maps API[11] offers a wide range of services, among those we will state the ones that are more relevant for our work, which are satellite imagery, street maps and route planning, for traveling by foot, car, bicycle, or public transportation.

The Google Maps Places service, which is used to search for places (defined in the API as establishments, geographic locations, or prominent points of interest) contained within a defined area, such as the bounds of a map, or around a fixed point. It also offers an autocomplete feature which is used for the type-ahead-search behavior of the Google Maps search field. When a user starts typing an address, autocomplete will fill in the rest.

The Google Maps Directions service, which is used to calculate directions (using a variety of methods of transportation) has some limitations in case we use the standard plan (free).

In short we can use Google Maps for the following purposes: to find places, to use the autocomplete functionality, to calculate routes and to present the final result to the user.

Our work requires that the users should be authenticated in order to be able to use it. An authenticated user is defined as someone that is enrolled in the centralized authentication system of the organization, in this case Instituto Superior Técnico (IST).

We want to avoid asking for the IST credentials, so that our system doesn't have to manage passwords, instead we want to delegate the authentication to another service, we can even ask IST for access to only specific user data that our system needs, this way if our system becomes compromised the IST credentials are still safe. Nowadays this delegation of authentication is the most used method to authenticate a user.

We analyzed several frameworks related to this feature: **OAuth 2.0**[12], **OpenID**, **OpenID Connect**[13] and **CAS**[14]. The most relevant being OAuth 2.0 and CAS.

OAuth 2.0 is the next evolution of the OAuth protocol which replaces and obsoletes the OAuth protocol[15]. This version focuses on simplifying client development while providing specific authorization flows for web applications, desktop applications, mobile phones and living room devices. The key aspect here is that OAuth 2.0 is not an authentication protocol [16]. Its focus is authorization, it grants access to data, functionality or other things without having to deal with the authentication.

Central Authentication Service (CAS), is a single sign-on (SSO) protocol which offers some key features like support for standards (OAuth Protocol, OpenID & OpenID Connect Protocol), RESTful API, support for clustering, services management and long term SSO. The highlight is that it allows web applications to authenticate users without gaining access to a user's security credentials, such as a password, which is exactly what we are looking for. Although we referred to CAS as a protocol[17] we can also refer to the software package that implements this protocol.

We analyzed two very well known technologies for databases, **MySQL** and **MongoDB** and compared them both. We concluded that MongoDB is a more flexible, scalable and available system, with a quicker query response. While MySQL allows more complex and multi-row transactions protected by the security and data integrity offered by relational databases. Each one excels in certain scenarios and is unfit for others. Due to their very different structures each one contains features not found in the other.

3. Proposed approach

In this section we detailed the requirements and functional aspects of both our systems, the Carpooling and the Mobility Portal.

3.1. Carpooling

The main goal of our Carpooling service is to allow driver users to offer trips with a specific departure date, duration, starting and ending locations, with a given number of seats and price per seat. Passenger users can perform queries on such trips (e.g. retrieve all trips that go from point A to B and depart at 01/11/2018 at 15:00) and book seats by issuing a request that can be accepted or rejected by the corresponding driver. In order to maintain the system we created an administrator role.

Our system has three different roles for users:

- Passenger;
- Driver;
- Administrator.

We will list below all the possible operations that our system allows:

- As a Passenger
  - Find a trip;

  - Request to book seats on a trip;
  - View requests;
  - Cancel requests;
  - View current and past trips;
  - Rate trips.
- As a Driver
  - Create a trip;
  - View book requests;
  - Accept/reject requests;
  - View current and past trips;
  - Delete a future trip;
  - Delete recurring trips.
- As an Administrator
  - Add/remove preferences;
  - Add/remove roles;
  - Add administrator permissions to a user.
- As any role (Administrator/Driver/Passenger)
  - View notifications;
  - View public profile;
  - Edit profile.

A very important aspect of any carpooling service is trust between users. Our system targets organizations where, usually, there is a good level of trust between users. Most often such organizations rely on a central authentication system. Therefore our system was designed to easily interface with such a system. In our test case we are using Fénix for authentication as well as to extract some basic information about the user.

In order for the users to have a clear understanding of our system and how to interact with it as easily as possible we decided to divide it into three main menus, which are displayed on a navigation bar on the top of the page, each of these menus have their own set of tabs:

- Passenger
  - Find Trip;
  - Requests;
  - My Trips.
- Driver
  - Create Trip;
  - Requests;
  - My Trips.
- Profile
  - Notifications;
  - Edit Profile;
  - Public Profile.

The roles that a user can have fluctuate as he navigates the interface of our system, more precisely the navigation bar with the menus of a passenger, driver and profile (which is generic). The typical user will only jump between passenger and driver. The administrator role is protected with a special login (username and password) that is required in order for a user to assume that role, which is in the same interface that enables the login with Fénix. If a user navigates to any of the sub-menus of the passenger then he is currently assuming the role of a passenger, or if a user navigates to any of the sub-menus of the driver then he is currently assuming the role of a driver. All the sub-menus of each menu are grouped in a drop-down list on the top right of the interface.

3.2. Mobility Portal

The Mobility Portal targets organizations where, usually, there is a good level of trust between users. It leverages all available transportation methods of such organization by creating plugins able to extract routes with schedules from the systems that handle those transportation methods. The main goal of the Mobility Portal is to allow a user to submit a request to search for trips from an origin point to a destination point with a certain departure date. This request is then process by the system which will use every available plugin to obtain possible routes that fulfill the request. The routes obtained may not fulfill the user request, in that case the system tries to find additional routes that can connect to the ones already received (e.g. walking 500m to be able to take the carpool).

We analyzed how the Mobility Portal could communicate with external services to get route information and their schedules. In order to have a modular solution which was a very important requirement to be considered in our system's design, we decided that our system would communicate with external services through plugins. This plugin architecture enables our system to be easily extended to collect information from many different services by adding new plugins that can query those services. The main external service that the Mobility Portal queries is our Carpooling service, although in the future more options can be added, such as a bike-sharing services and the IST Shuttle (which handles transportation between campi, Alameda and Taguspark).

We then moved to the algorithmic part of our system, more precisely to how our system will compute a result for a given input. After reviewing how related approaches handled this part we decided to implement the logic in the following way:

- To a user input to search for trips (origin, des-

tination and departure date) the system takes the departure date and queries every available system through its plugins to get every trip on that day;

- If after asking all plugins for information the system does not get any results, it will delegate that task to a Google Maps Plugin. This Plugin is the last resort to complete what is left of our current task;
- Else the system got some results, in case the origin and destination of those trips are a direct match with the ones given by the user the system does not need to do any more processing. Otherwise the system takes the points that do not match (origin and/or destination) and tries to connect them to the ones given by the user with routes from Google Maps.

We also analyzed the best way to present the results of a query to the user, such that he can easily understand them. Since the results are mainly routes and schedules we believe that the best way to understand them is with a list of possible routes and their respective schedules, from which the user can then select one and see it on a map. He can also follow a link present on each trip to book them.

We designed a simple interface for the Mobility Portal. Just like the Carpooling interface, we offer the Google maps auto-complete functionality to help choosing real locations for the origin (pick-up) and destination (drop-off) points, as well as the departure date that can be either written or chosen from a widget.

After the user has filled all the required parameters, he can then press the ”Search” button to submit the request. The results will then be displayed on a list below, and are organized by:

- Name of the service that answered with available routes;
- List of answers, which can be just one trip, or a series of trips that connect each other in a way that there is one trip that starts on the desired origin and one that ends on the desired destination.

Each trip of that list can be clicked which will display the route on the map. There is also a link to that trip which will redirect the user to the information of that trip on the respective service (e.g. go to the Carpooling ”View trip” interface which allows the submission of booking requests).

4. Implementation

In this chapter we detail the most relevant aspects pertaining the implementation of both the Carpooling and the Mobility Portal systems. We start by briefly describing the architecture of each system.

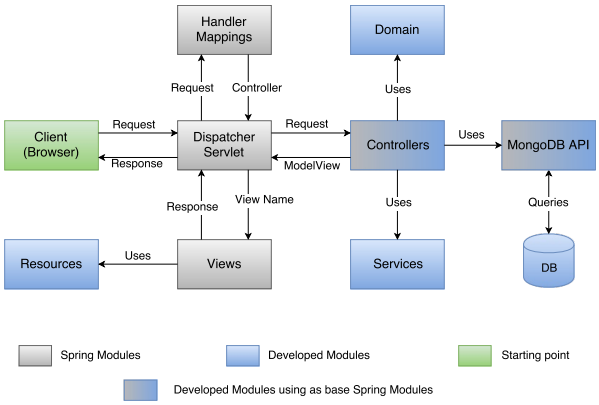

Figure 1: Carpooling system architecture

4.1. Carpooling

The main software modules of the carpooling system, as well as their most relevant interactions, are illustrated in Figure 1.

The grey modules are already defined/implemented by Spring and are used by the Spring engine without any need for customization. The blue modules were either developed or customized in the context of this work. Although the Controllers and the MongoDB API are also part of Spring they needed to be customized so that they could use the other developed modules (Domain, Services and the Database).

The Carpooling system is implemented in Java using Spring MVC (introduced in Section 2) which handles HTTP requests and returns the appropriate HTML content. Before this content is returned to the user (by Spring MVC) we use Thymeleaf to perform a server-side rendering of the HTML which enables us to easily fill an HTML page with dynamic content from the server. After all the dynamic content is inserted, Thymeleaf returns the HTML to the Spring MVC control which then returns it to the user (more precisely, to its browser).

The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements:

- The **Model** which encapsulates the application data;
- The **View** which is responsible for rendering the Model data and, in general, generate HTML output that the client (browser) can interpret;
- The **Controller** which is responsible for processing user requests, building an appropriate Model and pass it to the View for rendering.

Recapping what was introduced in Section 2 we

describe the Spring MVC modules of the Carpooling system architecture:

- **DispatcherServlet** is the front controller of the framework and is responsible for delegating control to the various modules during the execution phases of an HTTP request;
- **HandlerMapping** selects which Controller handles each incoming request;
- **Controller** interfaces between the Model and the View to manage incoming requests, gets processed data from the Model and advances that data to the View for rendering;
- **Views** encapsulate both the **ViewResolver** and the **View**
  - **ViewResolver** selects a View based on a logical name for the view;
  - **View** is responsible for returning a response to the client. Some requests may go straight to view without going to the model part; others may go through all three.
- **MongoDB API** provides an interface for querying the database;
- **Domain** defines the system classes and what information they hold in a structured fashion;
- **Services** have the logic that is not handled by the Controllers;
- **Resources** is responsible for the templates used to build the HTML response to the user;
- **DB** is the database that will store the system data.

The typical flow of information would be starting on the module **Client (Browser)** which is where the HTTP requests that are sent to our system are generated as a result of the interaction of the user with the interfaces provided by the system. The **Dispatcher Servlet** and the **Handler Mappings** are handled by the Spring engine, and their function is to find the adequate Controller to process the current request. The **Controllers** then use several modules like the **MongoDB API** to query the **Database** to get or save **Domain** objects. The **Services** to use their logic in order to process the request. The **Views** module will then find the requested HTML page on the **Resources** module, perform the necessary rendering of the HTML with Thymeleaf and return it to the Client.

The Carpooling system has two different types of communication, server-server and server-client(browser):

- **Server-server**: this type of communication is conducted between our server and an external server (e.g. Mobility Portal), since the data will only be used by servers it does not need to be presented with a user-friendly interface, thus the communication is done with a basic REST API (Spring) and the information is encoded in JSON format/XML, which is a structured human-readable format, but not very friendly for the user;
- **Server-client(browser)**: this type of communication is done between our server and a browser, since here there is user interaction the data needs to be presented in a user-friendly interface, thus the communication is done using Spring MVC and Thymeleaf.

The key difference between a traditional Spring MVC controller (server-client) and a RESTful web service controller (server-server) is the way the HTTP response body is created. While the traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data, which is written directly to the HTTP response as JSON/XML.

4.2. Mobility Portal

The main goal of the Mobility Portal is to allow a user to submit a query to search for trips from an origin point to a destination point with a certain departure date. This query is then process by the system which will communicate with several other external systems and obtain possible routes that fulfill the request. This communication is performed through modular plugins that must be developed and deployed to enable interfacing with each external system. Currently the Mobility Portal only communicates with our Carpooling system.

The Mobility Portal architecture has a similar architecture to that of the Carpooling system (described on Section 4.1) since the underlying technologies are the same. One difference, is that the Mobility Portal does not use a Database and thus the MongoDB API is not used either. A high-level view of the architecture of the system is illustrated in Figure 2, where the ”Spring MVC” module is an abstraction of the Spring MVC framework.

5. Evaluation

As with every software system, or any other kind of system for that matter, it is important to test it with a sample of target users before deploying it to assess how well it performs as well as to detect major flaws and be able to correct them before release to a larger audience. In the beginning of this document we described a series of requirements that

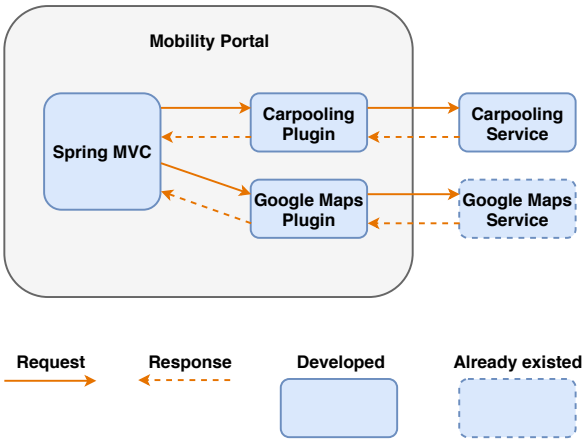

Figure 2: Mobility Portal top level architecture

our final solution should comply with. These requirements were addressed by features that need to be tested so that we can understand if they work as intended, to what extent they are easy to use and if there are any improvements that can be implemented. With this in mind we decided to conduct initial testing (also called Alpha Testing) with a few users when the Carpooling system was 70% - 90% complete, and later on some Beta Testing when the Carpooling system was 90% - 95% complete [18].

The main objective of the Alpha testing stage was to check whether we were on the right path, if there were any bugs that went undetected and if there were features that could be further improved. This proved to be very helpful since we were able to identify several problems and thus improve the system still during the main development.

The main goal of the Beta testing stage was to grasp if users, in practice, understood the interface of the Carpooling system, were able to navigate/use it easily, measure user satisfaction and collect additional suggestions. In this stage, a set of users were given the system to follow a script with a series of predefined tasks that were meant to cover most of the interactions that a user can have with the system. For each task the user was asked to rate it in terms of difficulty from 1 (Hard) to 5 (Easy) and comment on its use. In total, 12 users tested the system, which according to a study from Faulkner (2003) [19], the mean percentage of problems found relative to the number of participants (12 users) is between 94% and 97%, with the minimum being between 82% and 90%, thus suggesting that our testing pool of users can provide valid and significant results. Each user took in average 14 minutes to complete the testing, although not all provided an answer to the open questions (which were optional), the results were positive in general. Analyzing the average difficulty for each task, shown in the chart of Figure 3, we can conclude that the Carpooling system had a very good feedback from the users, with an average difficulty rating of 4 or more in each task.

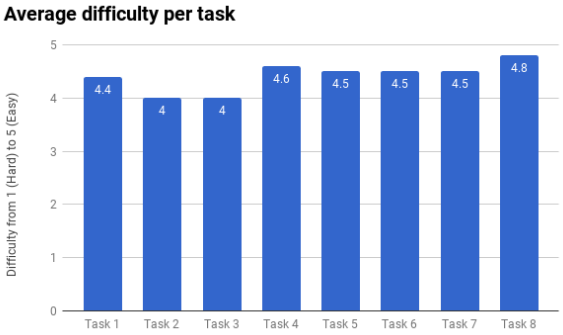

Figure 3: Average difficulty per task

Beta testing has shown us that after the initial learning curve the Carpoling interface was easily manipulated and understood, and that users could easily find and create trips in little time. Thus achieving the main objective of *Build our own Carpooling service and add the respective plugin to our Mobility Portal that can query that service*.

6. Conclusions

With all the transportation options available nowadays it becomes difficult to search for the most convenient option to travel from one place to another. As we have seen in this work there is no particular system that allows for example, the college students, to search for transportation options that only their college provides and only they can use, while at the same time searching for general transportation options (e.g. public transports). This is, in general, true for most other large organizations with multiple campi, thus we found the need to develop a solution that allows an authenticated user from a given organization to search for the most convenient option to travel from one place to another using several different transportation services including the ones provided by the organization itself to its users (students, employees, affiliates, etc). The Mobility Portal targets organizations where, usually, there is a good level of trust between users. It leverages all available transportation methods of such organization by creating plugins able to extract routes with schedules from the systems that handle those transportation methods. Since this system aims to be deployed on any organization we thought it would be interesting to develop a Carpooling system and use it as one of the external services of the Mobility Portal.

To achieve this we first identified the different inputs necessary to both our systems and how to communicate with the external services to get route information and their schedules, thus we ex-

plored how other systems accomplish this. After analyzing this we defined all the information that users would have to handle and in order to have a modular solution we decided that the Mobility Portal would communicate with external services through plugins (including our Carpooling service). This plugin architecture enables our system to be easily extended to collect information from many different systems by adding new plugins that can perform queries on such systems. We then moved to the algorithmic part of our system, more precisely to how both our systems would compute a result for a given input and display it to the user in an intuitive and useful way. In order to understand the best approach to use we analyzed several similar platforms, so that we could understand their advantages and disadvantages. After considering this we defined the Mobility Portal's behavior which is based on looping through plugins and the Carpooling behavior, since the results of both systems are mainly routes and schedules we decided that the best way for the user to understand their output would be with a list of possible routes and their respective schedules, which he can select and see on a map.

After the Carpooling system was 70% - 90% complete we decided to run some basic tests with users (Alpha Testing) to see if we were on the right path, if there were any bugs that we did not cover and if there were features that could be improved. This testings proved very helpful as stated in section 5. Then when the Carpooling system was 90% - 95% complete we did some Beta Testing which had the purpose of checking if users, in practice, understood the interface of our system, were able to navigate/use it easily, measure user satisfaction and collect suggestions.

Our initial goals of: *Build our own Carpooling service and add the respective plugin to our Mobility Portal that can query that service* and *Create a modular system (Mobility Portal) that allows users to define an origin, destination and departure date and search for different ways to travel from one point to the other on that departure date* were achieved, as our evaluation shows in section 5 (although the Mobility Portal was not tested with real users).

The systems truly work, despite having a few minor flaws to be improved in the future.

References

[1] Amsterdam Arena Mobility Portal. [Online]. Available: http://www.amsterdamarena.nl/stadium-surroundings/arena-mobility-portal.htm

[2] 511 SF Bay. [Online]. Available: https://511.org/

[3] Boleia.net. [Online]. Available: http://www.boleia.net/

[4] Blablacar. [Online]. Available: https://www.blablacar.pt/

[5] Poparide. [Online]. Available: https://www.poparide.com/

[6] SINGU Car Pooling. [Online]. Available: https://carpooling.tecnico.ulisboa.pt

[7] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[8] Spring MVC. [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[9] Jakarta Struts Web framework. [Online]. Available: http://struts.apache.org

[10] Thymeleaf. [Online]. Available: http://www.thymeleaf.org

[11] "What is the Google Maps API?". [Online]. Available: https://developers.google.com/maps/

[12] The OAuth 2.0 Authorization Framework (RFC 6749).

[13] "OpenID Connect". OpenID Foundation.

[14] "JASIG CAS Protocol Page". Apereo/JASIG.

[15] The OAuth 1.0 Protocol (RFC 5849).

[16] "User Authentication with OAuth 2.0". OAuth.net.

[17] Drew Mazurek. "CAS Protocol Specification 3.0".

[18] Alpha Testing and Beta Testing (A Complete Guide). [Online]. Available: http://www.softwaretestinghelp.com/what-is-alpha-testing-beta-testing/

[19] Laura Faulkner (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing.