# Social Routing

## Project and Seminar

Instituto Superior de Engenharia de Lisboa

*Authors:*
Baltasar Brito
email: baltasar.brito@gmail.com
phone: 915953552
Bernardo Costa
email: bjmcosta97@gmail.com
phone: 913897555

*Supervisor:*
Pedro Félix
email: pedrofelix@cc.isel.ipl.pt

April 29, 2019

## Contents

**1 Introduction** **3**

**2 System Structure** **4**

**3 Client Application** **5**

**4 Social Routing Service** **6**

4.1 Social Routing API . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

4.2 Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.3 Database Management System . . . . . . . . . . . . . . . . . . . . . 10

**5 Updated Timeline** **11**

**6 Risks** **12**

**7 Conclusion** **13**

# 1 Introduction

## Report Structure

The structure of the report is based on the previously made timeline. Each of the timeline's points is explained and evaluated with regards to it's completion and difficulties that were overcome in it's implementation.

The previously made timeline had incorrect delivery dates and as such this report will also present a correct and improved version of the future timeline to follow as well as the risks it might contain.

## 2 System Structure

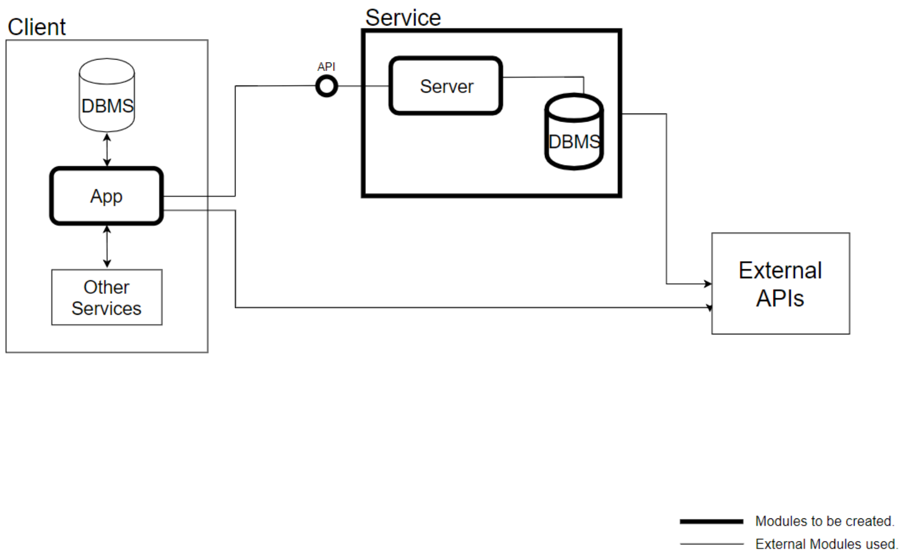


Figure 1: System structure.

## 3 Client Application

### Infrastructure Design

### Route Creation

### Communication with the Social Routing API

# 4 Social Routing Service

The Social Routing Service is responsible for providing access to data and functionalities to a client through an an HTTP [2] based API. It receives HTTP made requests with certain specifications and responds with resources generated by the the server. The following diagram details the building blocks of the service:

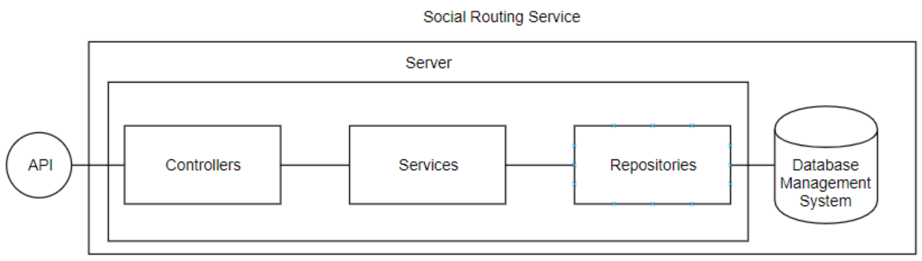


Figure 2: Social Routing Service structure.

## 4.1 Social Routing API

The goal of the Social Routing API is to expose the service's functionalities to a client, in this case the an Android[1] based client application. The API should be used to store or retrieve routes and user information, as well as to find the closest route available to a specific user or location.

It was built in the Kotlin [9] programming language and using the Spring MVC framework [16]. The build system is that of the server, which they both share with the framework itself, Gradle [3].

All API access is done over Hypertext Transfer Protocol and the API data is sent and received in the JSON [7] format. The exception is on error responses which follows the JSON+problem [10] standard.

Currently the API supports requests in a limited number of HTTP *verbs* (GET, POST, PUT, DELETE) [14], and has a functional but incomplete error handling system. Some request specifications are already complete, such as the media type desired in a POST request but some are not such as user authentication.

### Endpoint for Route Creation

The first task when building the API was to provide a way for a Route to be created because of the considerations it requires. Namely, the characterization of a Route in it's data form and the metadata it needs in order to be stored and retrieved.

This goal was achieved with success [13] but with some minor problems in the characterization aspect occurring from time to time. Initially a route was characterized by a set of coordinate pairs, that each had the latitude and longitude of a given geographic point, but as the project grew the metadata required to characterize a route grew with it. At this time, for a route to be created it requires not only the set of coordinates but also its creator's information and its assigned categories.

### Finalization

After the the API's structure was complete with the creation of a route the rest of the endpoints were added and are now functioning properly [15]. There are some functionalities left that will only be complete later on, like user authentication and error handling, the last one being now done but in an incomplete albeit functional way.

When adding each of the endpoints to the API the difficulty relied in a separation of concerns and requirements of both the server side and client side modules.

## 4.2 Server

The role of the server within the system is to receive data, transform it, and either store it or process it and return it as a response. The technologies it requires to function are the same as the API.

The *flow* of an HTTP request that arrives on the server goes through the following pipeline: a Controller is assigned to a specific endpoint which will then use at least one service. If the request requires stored data the service will request it from a repository, if not, the service processes the data and returns it in the correct form back to the controller which responds to the request with a response in a well defined format, depending on the request made.

### Communication with the Database Management System

Initially the communication with the database was made using Spring Data JPA [6] but after the initial implementation with it, it revealed itself to be time consuming on the learning side, and limited in the management of the database connection and SQL queries desired. That caused a switch to a different approach, using JDBI [5]. JDBI [5] is an API which is built on top of the JDBC driver [4].

and is used directly in the server by the Repository component to communicate with the database.

### Infrastructure Design and Implementation

The structure of the server follows the one showed in figure 1. Two Controllers were built, one for user related requests and other for route related ones. Following the diagram, the services follow the same logic, one for route related processing and other one for user processing. The final layer is the repositories which depends directly on the entities present on the database. There are two repositories as well, one for user related operations and another for those that are route related.

An also very important part of the server is also the conversion of data. Currently, a received request is transformed from received data in the JSON format to an input type object which is received by a controller, then to a data transfer object and sent to one or multiple services and after that to a domain object which might be sent to a repository if database storage or retrieval is necessary.

Setting this goal so early in the timeline allowed us to establish a foundation to build on top of in a very quick fashion, being that most of the work was done decided initially the rest was just a question of adding an extra Repository, Service or Controller as well as the required type converters.

### Add All Functionalities Except Search

The functionalities of the server required in this point were all the ones considered to support a functional API and as such, after the initial structure was done, most of the requirements that followed were a matter of adding and extra component horizontally being that the vertical structure was already complete. As far as the search functionality goes, it is implemented in a minimalistic version, allowing for rout searching by location only.

## 4.3 Database Management System

The database management system chosen was PostgreSQL [11], using the hybrid functionality of storing valid JSON directly in a table field [8]. The database is used to store all entities required for the service to function and to deliver them to repositories in need. The decision of choosing JSON as a type to store data comes with the need of storing large sets of coordinates belonging to a single entity, this will allow us to make faster and easier calculations of times and distances between routes and points rather than if a Point was it's own entity.

The following entity diagram represents the structure of the database:

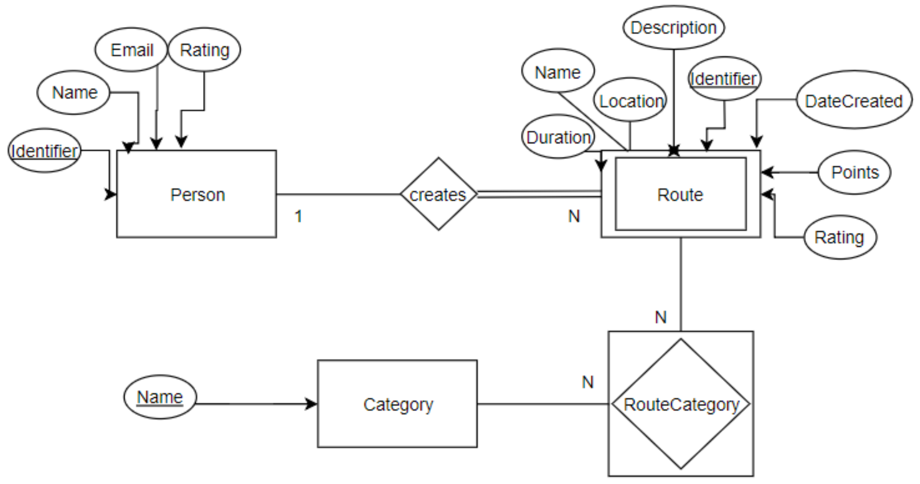


Figure 3: Entity Relationship Diagram.

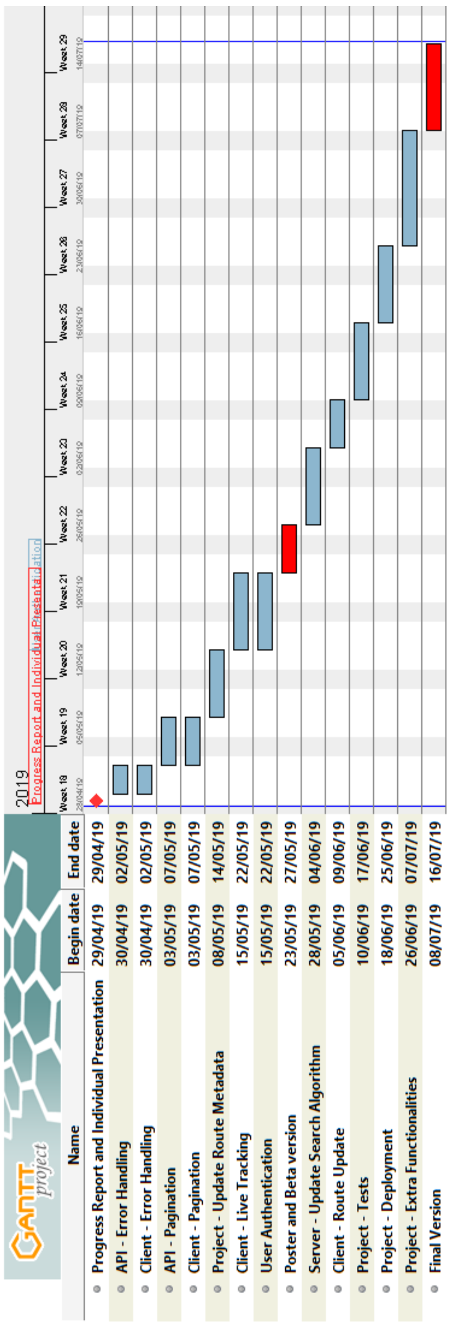
### Route Saving

Route saving, like the first points of each other component, was chosen to be able to have a well defined structure of the whole project as quickly as possible with the goals of structure first, implementations later. After characterizing a route [12] the project had a fully functional minimalistic structure.

### Remaining Entities

After the initial structure was built the remaining entities were added to the database as needed. To maintain integrity and consistency of the data functions were added to support more complete queries that manipulate multiple tables. When inserting a route for example, the route has to reference at least one category, and as such an insert to the route table has to guarantee an insert to the RouteCategory table. The procedures are called by the repositories and executed by the database.

## 5 Updated Timeline



| Name | Begin date | End date |
|---|---|---|
| Progress Report and Individual Presentation | 29/04/19 | 29/04/19 |
| API - Error Handling | 30/04/19 | 02/05/19 |
| Client - Error Handling | 30/04/19 | 02/05/19 |
| API - Pagination | 03/05/19 | 07/05/19 |
| Client - Pagination | 03/05/19 | 07/05/19 |
| Project - Update Route Metadata | 08/05/19 | 14/05/19 |
| Client - Live Tracking | 15/05/19 | 22/05/19 |
| User Authentication | 15/05/19 | 22/05/19 |
| Poster and Beta version | 23/05/19 | 27/05/19 |
| Server - Update Search Algorithm | 28/05/19 | 04/06/19 |
| Client - Route Update | 05/06/19 | 09/06/19 |
| Project - Tests | 10/06/19 | 17/06/19 |
| Project - Deployment | 18/06/19 | 25/06/19 |
| Project - Extra Functionalities | 26/06/19 | 07/07/19 |
| Final Version | 08/07/19 | 16/07/19 |

Figure 4: Updated timeline.

## 6 Risks

The main risk while making this project is the time to complete it and the way to minimize this is a well thought plan of action, hence the restructuring of the original timeline. The following list of risks will help us manage the time left as well as give us an idea of were we stand in the long term planning.

- Search Algorithm: given search parameters each user should have the best possible route suggested to him.
- User authentication: using an external service and integrate it with both the client application and the Social Routing Service.
- Live tracking: the user should be able to follow the route in real time, on his device.
- External APIs comprehension and usage might take more time than initially expected.
- Tests: testing each component correctly might involve using external unknown libraries and the time necessary to create them might not be enough.

## 7 Conclusion

## References

[1] *Android Documentation.* URL: `https://developer.android.com/guide/`. (accessed: 29.04.2019).

[2] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1.* URL: `https://tools.ietf.org/html/rfc2616?spm=5176.doc32013.2.3.Aimyd7`. (accessed: 29.04.2019).

[3] *Gradle Documentation.* URL: `https://docs.gradle.org/current/userguide/userguide.html`. (accessed: 29.04.2019).

[4] *JDBC driver definition.* URL: `https://en.wikipedia.org/wiki/JDBC_driver`. (accessed: 29.04.2019).

[5] *JDBI Documentation.* URL: `http://jdbi.org/`. (accessed: 29.04.2019).

[6] *JPA Documentation.* URL: `https://spring.io/projects/spring-data-jpa`. (accessed: 29.04.2019).

[7] *Json Documentation.* URL: `https://www.json.org`. (accessed: 29.04.2019).

[8] *JSON PostgreSQL Documentation.* URL: `https://www.postgresql.org/docs/9.3/functions-json.html`. (accessed: 29.04.2019).

[9] *Kotlin Documentation.* URL: `https://kotlinlang.org/docs/reference`. (accessed: 29.04.2019).

[10] M. Nottingham, Akamai, and E. Wilde. *Json+problem Documentation.* URL: `https://tools.ietf.org/html/rfc7807`. (accessed: 29.04.2019).

[11] *PostgreSQL Documentation.* URL: `https://www.postgresql.org/files/documentation/pdf/11/postgresql-11-A4.pdf`. (accessed: 29.04.2019).

[12] *Route Characterization.* (accessed: 29.04.2019).

[13] *Route Creation Documentation.* URL: `https://github.com/baltasarb/social-routing/wiki/Social-Routing-API#create-route`. (accessed: 29.04.2019).

[14] *Social Routing API - HTTP Verbs Documentation.* URL: `https://github.com/baltasarb/social-routing/wiki/Social-Routing-API#http-verbs`. (accessed: 29.04.2019).

[15] *Social Routing API Documentation.* URL: `https://github.com/baltasarb/social-routing/wiki/Social-Routing-API`. (accessed: 29.04.2019).

[16] *Spring MVC Documentation.* URL: `https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html`. (accessed: 29.04.2019).