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

This progress report provides information about the current state of the project, specifically about what is done until now and what will be done in the future, it details the future project goals and the risks that will be dealt with.

The structure of the report is based on the previously created 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 timeline points match with each of the report's sections.

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

The project still follows the same System Structure that was established initially in the proposal, which has two major components and a third exterior one, communicating with each other, separating concerns and business logic. The following figure illustrates the system's structure:

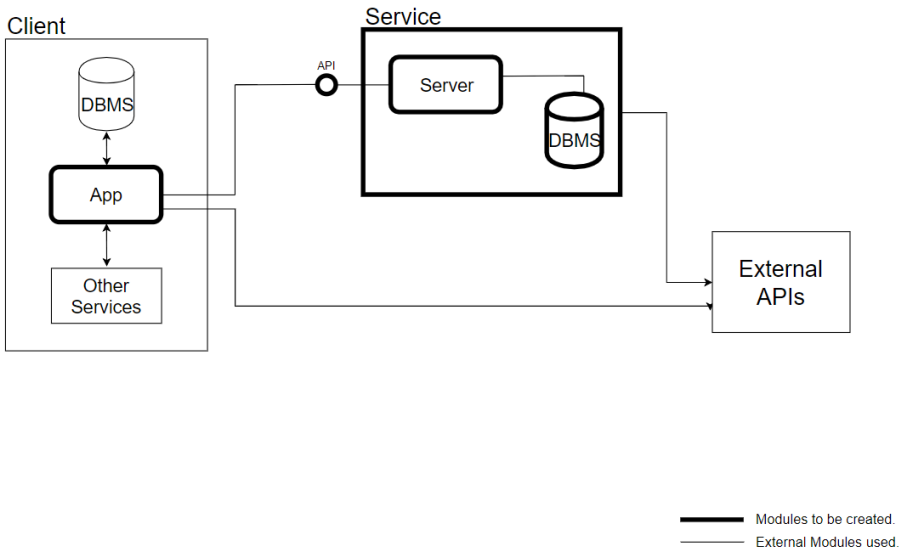

Figure 1: System structure.

3 Client Application

The first step was choosing the proper technology[3] for this component and Kotlin [13] seemed the best choice as far as supported programming languages of the Android Platform go, because it introduces functional features to support interoperability and intelligibility. The main reason Kotlin was chosen was to improve the coding experience in a way that was practical, fast, and effectual. After the technology was decided the definition of the Client application[2] Architectural principles followed, which is the core of this component. So the principles were the separation of concerns, where each module has its own purpose and that the user interface(UI) should be driven from a model, which in turn means that the view objects and the application components are independent of data handlers.

Route Creation

The Route Creation functionality is already done in the Client Application and is where the user can create its own Routes to show others, when searched. Initially the user is asked where is the location of the Route and then using the google maps API the map is obtained and it is zoomed to the chosen location. After that, the user can click on the map to select the Route points which are then connected by the application, forming a route. When the route is completed the user needs to fill a form to complete the metadata such as the name, description and category of the Route that is about to be created. The Route metadata could have some improvements in the future, to be more useful when searching for the correct Route.

Communication with the Social Routing API

To build communication with the Social Routing Service[5], the library Retrofit[16] was used, which is responsible for making Hypertext Transfer Protocol(HTTP) [6] requests.

Infrastructure Design

The infrastructure design[4] of the Client Application follows the principles described before and as such, it can be divided in four blocks: Activities and Fragments, View Models, Repository and the Remote Data Source. Where each has one objective in the Client Application:

- Activities and Fragments: contain logic that handles the UI and the actions related to operating system. These are the components that the user can interact with and obtain feedback about what is currently happening on the application.
- ViewModels : when the UI experiences a change, the ViewModel calls other components to load the data, and it can forward user requests to modify the data, however the ViewModel doesn't know about UI components, it is completely separated from them.
- Repository : handles data operations, knows where to get the data from and what API calls to make when data is updated. A repository can be considered to be a mediator between different data sources, such as web services.
- Remote Data Source: module that has the objective of communicating with external APIs and it communicates by doing requests to some Services, including the Social Routing API.

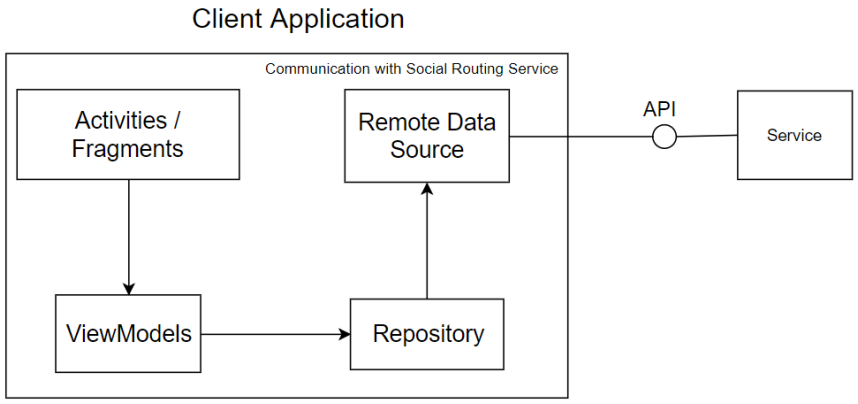

Figure 2: Client Application Structure.

4 Social Routing Service

The Social Routing Service is responsible for providing access to data and functionalities to a client through an an HTTP 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:

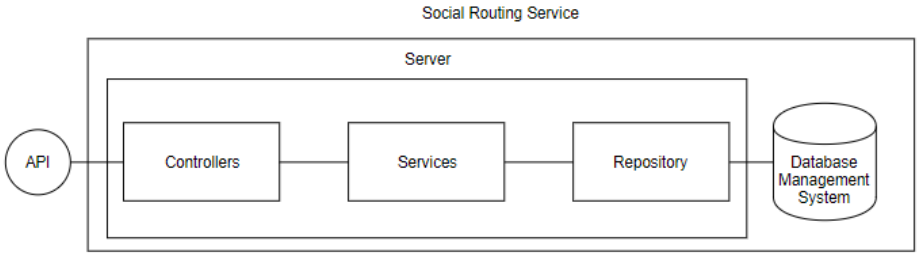

Figure 3: Social Routing Service structure.

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 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 programming language and uses the Spring MVC framework [20]. The build system is that of the server, which they both share with the framework itself, Gradle [7].

All API access is done over HTTP and the API data is sent and received in the JSON [11] format. The exception is on error responses which follows the JSON+problem [14] standard.

Currently the API supports requests in a limited number of HTTP *verbs* (GET, POST, PUT, DELETE) [18], 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 definition of a Route and the metadata it needs in order to be stored and retrieved.

This goal was achieved with success [17] but with some minor problems in the definition aspect occurring from time to time. Initially a route was 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 define 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 [19]. 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 might use a 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 [10] 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 [9] which is an API that is built on top of the JDBC driver [8] 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. A controller receives HTTP requests and forwards them to a service, which in turn will process the data, and if required will request or store persistent data through the use of a repository.

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.

Remaining 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 route searching by location only.

4.3 Database Management System

The database management system chosen was PostgreSQL [15], using the hybrid functionality of storing valid JSON directly in a table field [12]. 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 database entity.

The following entity diagram represents the structure of the database:

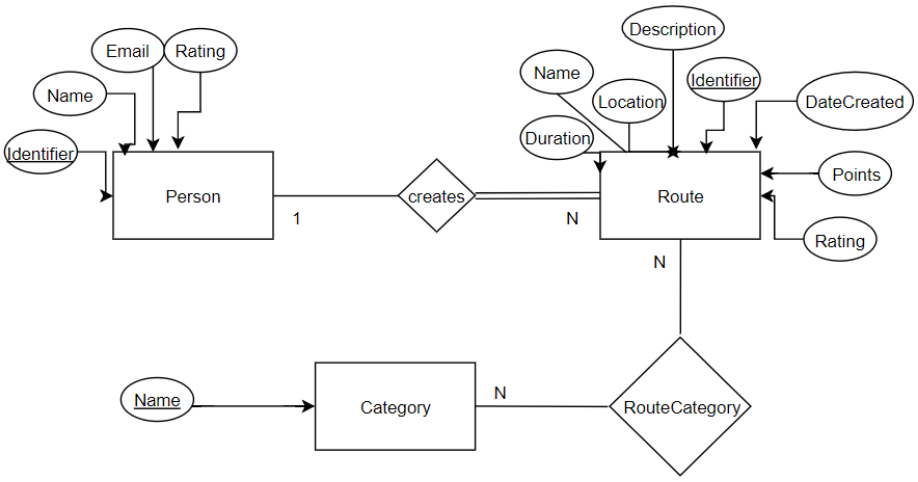

Figure 4: Entity Relationship Diagram.

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 defining a route 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

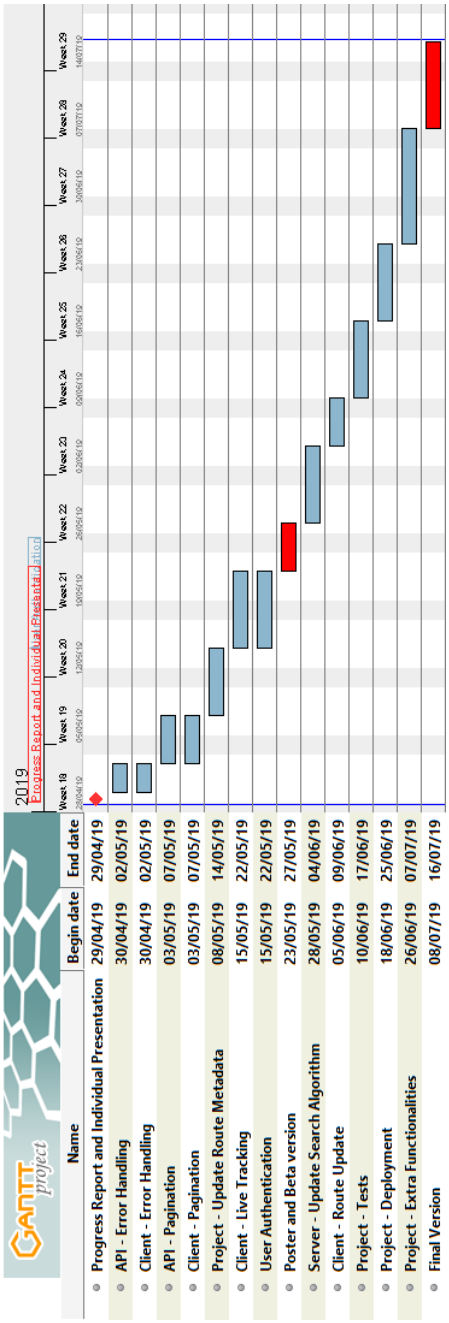

Figure 5: Updated timeline.

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

A major issue so far with the project was the incorrect planning of the first timeline. The dates for the deliverable milestones (progress report, beta and final version) were considered to be about one month earlier than the actual correct date and therefore the progress cannot be based solely on the original timeline. However, all the goals set for before the original delivery date of the progress report were accomplished and because of the extra time, some features planned only for after the delivery were also done. The tasks completed after the delivery were:

- Client: the route search functionality is already done, albeit in a minimalistic way. The user profile is also completed as well as route visual representation. All these tasks were in the API - finalize timeline task point.
- Social Routing Service: close to complete lacking only (complete) error handling, pagination and search algorithm improvement.

Considering this initial mistake in the original timeline creation, we created a new one to better suit the needs of the project's implementation. Creating a new timeline has some risks and as such we dedicated a chapter to list them. One of the first problems encountered was how a route should be defined. Several iterations were made before arriving at the current one. We avoid saying final because the definition of a route is susceptive to change when new functionalities are implemented. While implementing the Client Application some problems were encountered when utilizing the Google Maps API, namely the API key and the service's usage. Without an account with a credit card associated we were limited to one API call per day, which was limiting our testing capabilities. This was overcome by creating a wallet (offered by the service with an initial value). The process that is required to create a route was also a problem, there are some considerations to be had due to the nature of a path: where it starts, where it ends, if it is circular, if it is doable in a determined time amongst others. This topic is intertwined with route definition, changes in the definition also change the creation.

The Service had problems initially with the connection to the Database Management System, specifically because the choice of technologies was not the appropriate one, which forced a shift to a different one. This used valuable time because two different libraries were learned (Spring Data and JDBI) and later Spring Data was discarded.

References

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

[2] *Client Application Documentation*. URL: https://github.com/baltasarb/social-routing/wiki/Client-Application. (accessed: 29.04.2019).

[3] *Client Choice of Technologies Documentation*. URL: https://github.com/baltasarb/social-routing/wiki/Choice-Of-Technologies#client-application. (accessed: 29.04.2019).

[4] *Client Design and Implementation Documentation*. URL: https://github.com/baltasarb/social-routing/wiki/Client-Application#design-and-implementation. (accessed: 29.04.2019).

[5] *Communication with external APIs Documentation*. URL: https://github.com/baltasarb/social-routing/wiki/Client-Application#communication-with-external-apis. (accessed: 29.04.2019).

[6] 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).

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

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

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

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

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

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

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

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

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

[16] *Retrofit Documentation*. URL: http://square.github.io/retrofit/. (accessed: 29.04.2019).

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

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

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

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