Social Routing

Final Report - Project and Seminar

ISEL - Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Authors:
Baltasar Brito
email: baltasar.brito@gmail.com
number: 42466
Bernardo Costa
email: bjmcosta97@gmail.com
number: 42165

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

July 16, 2019

Abstract

This project consists of a mobile application to allow an application user to either create or undergo pedestrian routes. A route is a set of geographical coordinates (latitude and longitude) and as such a route is a path formed by sub paths on a map.

To application is structured in two major components, one with the front end, the Social Routing Client Application (SRCA) built for Android based devices and the back end, the Social Routing Service (SRS) which exposes it's functionality through an HTTP[16] API[9].

The SRCA is made for Android users, using Kotlin[33] as a programming language, and the guidelines made available by Google for Android application development. The features available are user login, profile viewing, route creation, searching, and live tracking. The live tracking feature helps the user while he is performing a route, informing the user about his current location and if he is far from the route. There is also the ability to rate other users route's.

The SRS exposes it's functionality through an HTTP API, and is subdivided into a Spring Application[56], using Kotlin as a programming language, and a Database Management System, using PostgreSQL[38]. The database is necessary because as a user creates routes, other users must be able to see them and as such storage is required for persistent information maintainability.

As the application requires users to access information such as routes created by other users as well as their own, it requires authentication. Both the SRCA and the SRS require the user to be authenticated, being that they communicate with each other, the user requests credentials from the SRCA that in turn reaches out to the SRS to get them. This is made with aid of an externa API being the SRS responsible for the generation of user credentials.

The major problems encountered were the route search functionality: a user has the ability to search for routes near his location, and authentication. The first was solved with aid from a PostgreSQL extension PostGIS[37] and the second with the Google Sign In API[25].

Contents

1 Introduction

This document describes the project's context, decisions and reasons behind them. It starts with the description of the global system architecture where a general system view is described. After that each component of the system is explained in greater detail. In this chapter the context of the application is explained as well as the features made available by the system that was built.

The project consists of system that provides the ability to define and share touristic pedestrian routes. It allows area exploration by utilizing user made routes as virtual tour guides to other users. Essentially a user of the Social Routing Application is be able to:

- Create any Route that is possible to represent on a map.
- Search user made routes and see them drawn on a map.
- Perform any chosen Route while being tracked by location in a live fashion.
- View his own or other user's profile.
- Rate routes based on his experience when undergoing such route.

In the context of the application, a route is a path from point A to point B, that goes through user selected sub paths that might either have relevance or simply provide the fastest way to the next point of interest of that route. An example of events when using the application might be:

A user at his hotel decides he wants to go sightseeing for an hour and check the surrounding area by foot.

- Ability to search routes made by other application users: the user starts the application and searches for a route specifying either his location or a desired one.
- Ability to filter results by paramaters such as route category or duration: The search parameter Cultural is chosen as a route category and the duration is set to short.
- The application suggests the first five routes available that match the user parameters.
- Ability to preview routes before making the choice to undergo one: The user selects one of the suggested routes and is shown the route on a map.

- Live tracking, the user is able to check his positioning on the map while performing the chosen route: the user chooses to begin the route and receives directions in real time on a map that he has to follow to undergo such route until it is done.
- Ability to rate other user's routes after performing them: The user finishes the route and evaluates it.

Apart from the mentioned features an application user also has the possibility of creating and editing his own routes as well as to check his profile.

2 System Architecture

This chapter goes into the detail about the system components and their connections as a whole. The system architecture organizes the system's necessities into manageable blocks as shown in figure 2.1. It is essentially divided into two major components, the Social Routing Client Application [13] and the Social Routing Service with a third one being the external services.

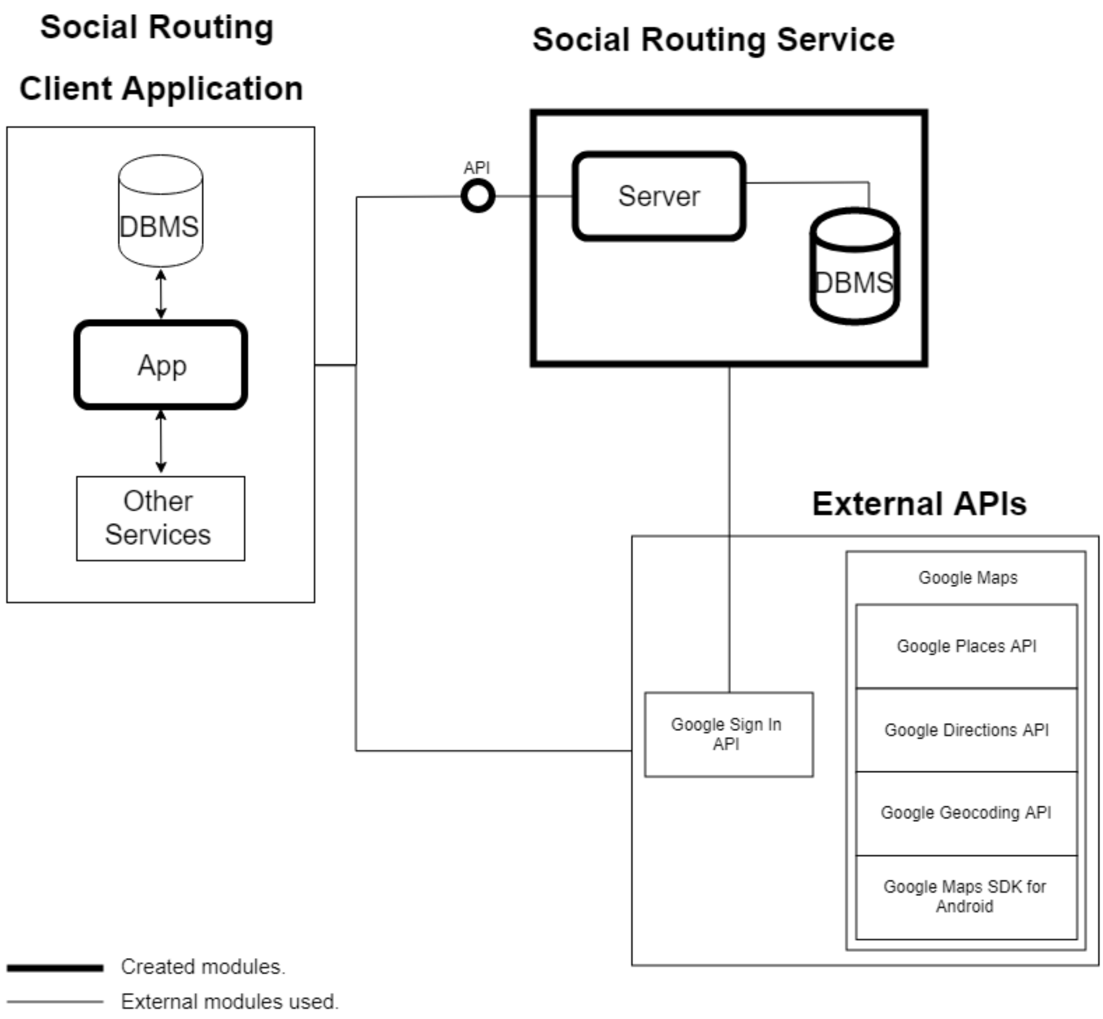

Figure 2.1: System structure.

The Social Routing Service is subdivided into two components, the Server and the Database Management System[15]. The Server exposes its functionality through an HTTP[16] API[9] (Social Routing API [49]) and as such, receives requests from a client, processes the received request and responds accordingly. The Database Management System is used to store the server's data, over which the necessary calculations are made. The Social Routing Service uses only one external API, Google Sign In[25], which is used to help with user authentication. to it.

The role of the Social Routing Client Application is to provide an interface to Android [2] devices which the user can interact with, to expose the project idea and essentially to demonstrate the Social Routing Service functionalities to the client. This component communicates with the Social Routing Service and external services when required, through the HTTP protocol. The external services are APIs provided by the Google Maps platform [22] to retrieve information such as the user profile from google accounts and content that helps managing the the Google maps, like locations, places and routes.

To authenticate a user in the system every component has a role, and because of it the general logic of authentication is part of the system's description. The figure 2.2 describes the authentication logic in the project's scope.

2.1 System authentication

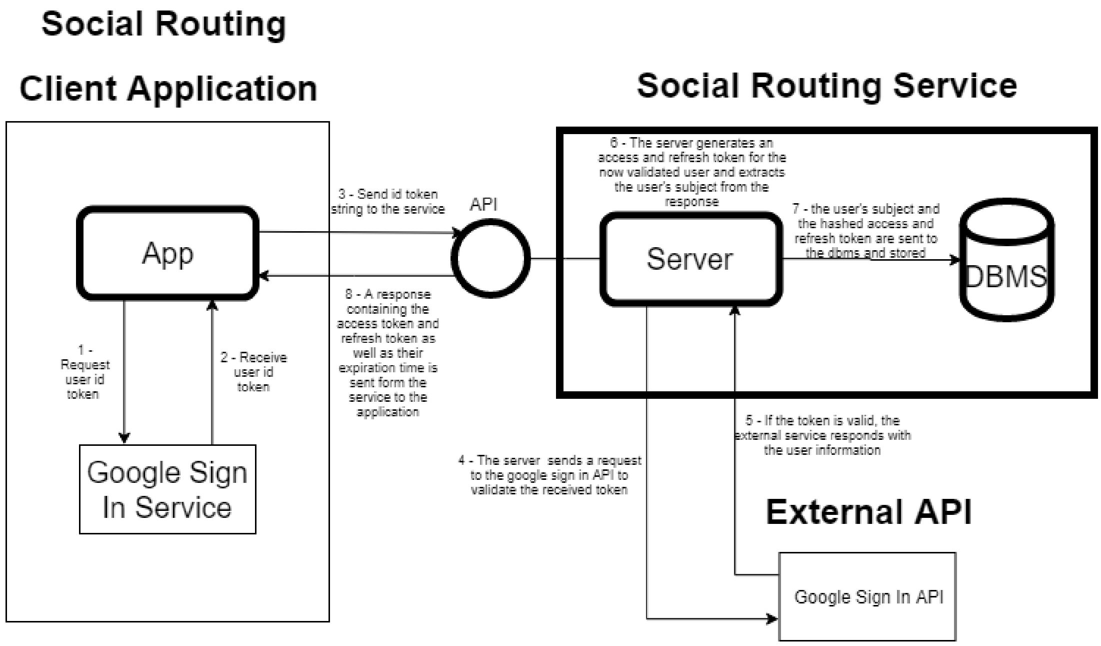

Figure 2.2: Authentication diagram.

The authentication process begins with a request from the Client Application for the user id token, obtained through a Google Sign In internal service. The received id token string is then sent in an HTTP Post request to the API, to try and obtain service authentication credentials. When the service receives the token it must verify it's authenticity and for that a request is made to an external API Google Sign In, that upon correct token validation responds with user related information of which the subject is part of. After the token validation is complete the server then generates an access and a refresh token, which it then hashes and sends to the database along with the retrieved subject. This information is stored to allow that in the future the user does not need to request credentials if they are already in his possession. After the storing process is complete a response is sent to the Client Application containing the generated access token and refresh token which will then be used to make authenticated requests to the server.

3 Social Routing Service

This chapter describes in detail the Social Routing Service and it's components, specifying the role of each of them in the system as well as their connection with each other.

The Social Routing Service (SRS) is comprised of three major components, a PostgreSQL based database to store user related information, a spring application to process data and an API to expose it's functionality. This logic is represented in the figure 3.1. Beyond the mentioned components it also uses the external API Google Sign In to aid with the user authentication process.

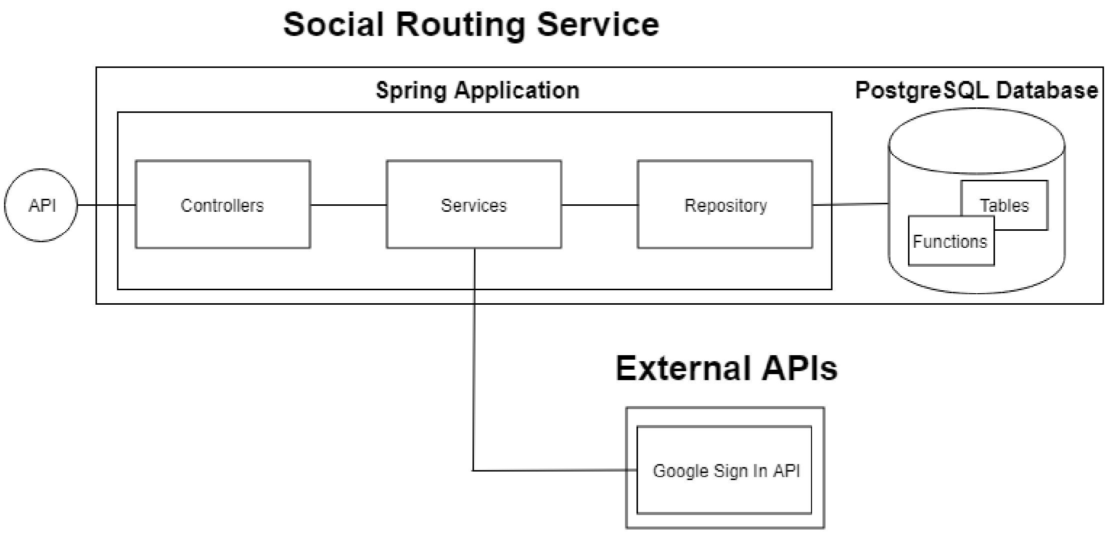

Figure 3.1: Social Routing Service architecture.

The components, although separated, must communicate with each other in order for the SRS to function properly. The API is exposed by the server using the HTTP protocol. The communication with external APIs made by the server is done through the HTTPS[27] protocol.

The communication with the DBMS is made using JDBI[30], a library built on top of the JDBC[29] driver. The main idea when building the communication with the DBMS was to guarantee that a single request made to the service would use a single connection to the DBMS. For that reason, the connection is both created and closed in the controller element of the system. This way it allows that the DBMS connection is used multiple times by the same request if required.

3.1 Data Model

The DBMS chosen was PostgreSQL[38], using the hybrid functionality of storing valid JSON[32] directly in a table field. 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 and this allows 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 main concept of the project is the creation of routes in a map and their retrieval according to certain parameters. To consider distance between a user and a route a possible search parameter, the PostGIS[37] PostgreSQL extension was used as it allows both cartesian and geodesic distances calculations.

3.1.1 Conceptual Model

The database entity diagram is shown in figure 3.2.

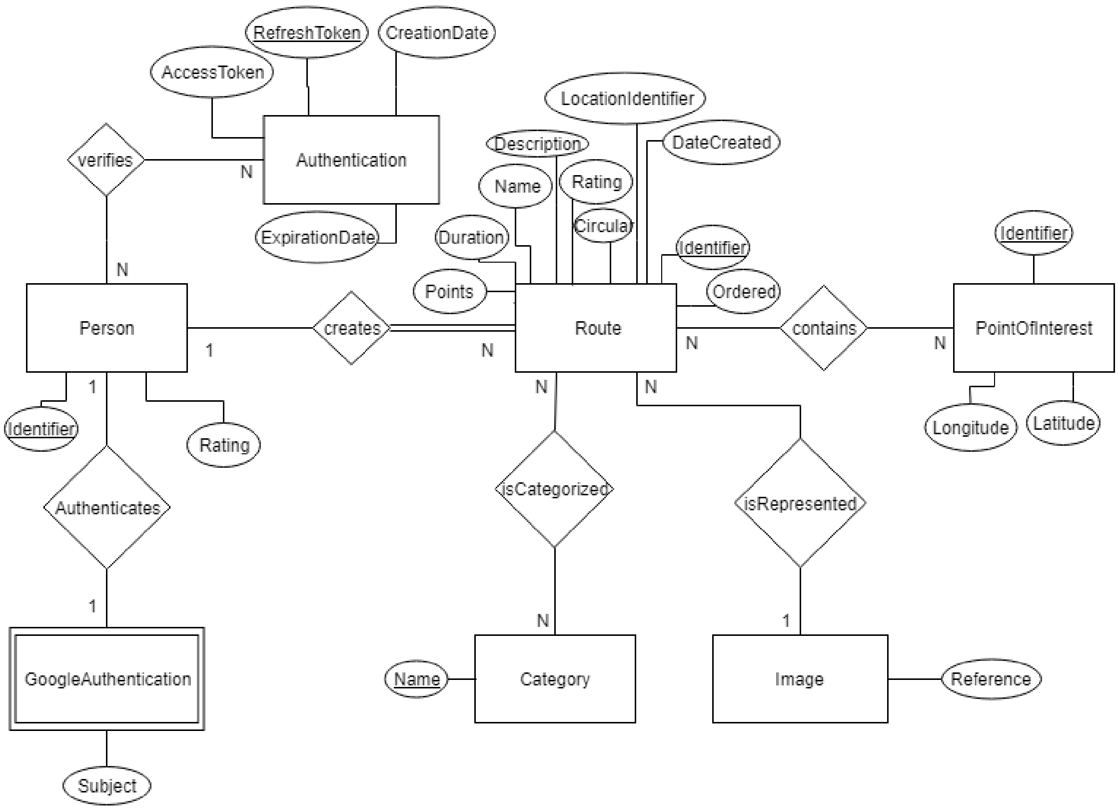

Figure 3.2: Entity Relationship Diagram.

Person

The entity Person represents a single user in the database.

Route

Entity that stores every information required to represent a Route. Special consideration was taken into the making of this entity as a Route must have large sets of coordinates, representing the path that it undergoes.

GoogleAuthentication

Google Authentication is the entity that stores the user unique identifier in the Google platform, the subject. In the future there will be multiple entities with different forms of authentication, hence the name GoogleAuthentication of the entity. It provides scalability to further augment the authentication process, which can be done with a new entity FacebookAuthentication for example.

Authentication

This entity is used to store authentication metadata regarding each database user. The tokens have an expiration time, and because of it their creation date is required as well as their expiration date.

Category

The category entity defines a route in a broader sense. It is used to filter the amount of routes being searched by a user. It can be one of the several: Sea, Sports, Cultural, amongst others. Each route must be assigned at least one category.

PointOfInterest

A route might contain geographic points of interest along its path, this is the entity that represents them.

Image

The image entity represents the image that is associated with the route, namely it's cover image. It is characterized by a reference that allows the image's URI to be fetched.

3.1.2 Physical Model

The physical model can be seen in a full in the database documentation[53], the following subsections discuss some of the table fields considered important.

Person

The table person has only two fields, the identifier of the person, generated by the database and the person's rating. A user can have multiple entries on the Authentication table to enable multiple device registration, but only a single Google Authentication entry as the google account is unique.

Route

The route table has amongst others, a field called points. This field is of type JSON, and is used to store sets of geographic coordinates. The field is a JSON array of point objects and has the following structure:

```
[
    {
        "latitude" : double,
        "longitude": double
    },
    ...
]
```

This structure allows the mapping of multiple points to a single route. Two other very important fields are the booleans ordered and circular. The field ordered is used to store if a route should start only at it's first geographic point or if the order of which a user can undergo is not defined. The circular parameter is used to verify if a route ends where it starts, because if that is the case, then the route can be started by the user in any of the route's geographic points-

GoogleAuthentication

The google authentication table is used to store google account information, more specifically the subject which is primary key. Only one google account per user is allowed.

Authentication

The authentication table stores the user credentials generated by the Social Routing Service. It contains the access token, the refresh token, the date of creation and the expiration date of the access token as well as the person identifier. The primary key of the table is the refresh token, as a single person can have multiple entries on this table, to allow multiple devices to be logged in while maintaining the same account.

Category

The category entity characterizes a route in a broader sense. It is used to filter the amount of routes being searched by a user. It can be one of the several: Sea, Sports, Cultural, amongst others. The categories available are manually inserted to the database. Each route must be assigned at least one category.

PointOfInterest

A point of interest is a pair of geographic coordinates (latitude and longitude) and it's reference to the google platform.

Image

The table image contains a single field which is it's primary key, a reference. This is the image's unique identifier in the google platform.

3.1.3 Route Search

The SRS allows that a service client searches routes that are inside a predetermined area. This feature is useful to determine if a route is close to a user location or not. This functionality is exposed in an SQL query[40] and an SQL function[39].

The route search algorithm is made in two different phases, the first uses a set of parameters such as the categories of the routes being searched, the duration desired and the geographic location being searched. These parameters function as initial filters to reduce the number of inputs being worked over.

After this initial list of routes is reduced to relevant routes, the algorithm must then check which of these routes are in the user desired area. The user desired area is by default a 1km circumference around the user's geographic coordinates, but a few other aspects have to be considered when choosing the perfect route. When a route is initially created, it has characterization related to its ordering, this allows for a better route suggestion algorithm, as a circular route can be started in any of its points, where an ordered route can only be started in it's first point. A user far from the starting point (over 1km distance) would not want it to be suggested to him, as he would have to walk too much to the starting point only to come back to undergo the route. An example of this scenario can be seen in 3.3

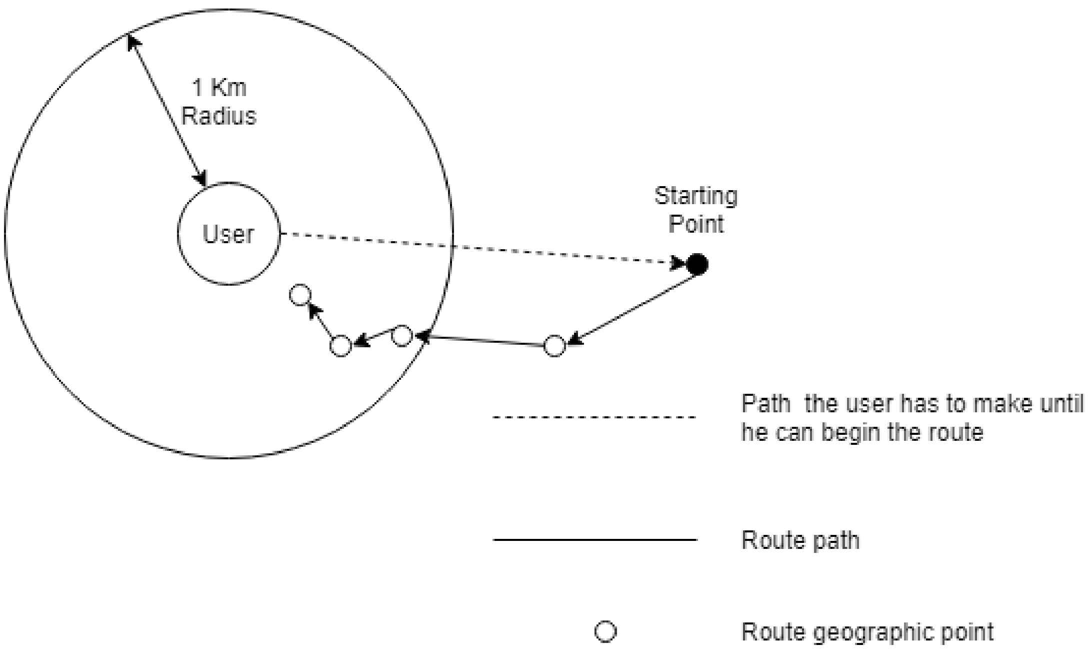

Figure 3.3: Starting point is outside the user area example.

For a route that starts where it also ends this poses no problem, as any point can be suggested as a starting point to the user as seen in 3.4 The user can go straight to the closest point and start the route, so the fact that the route is circular immediately makes it relevant to the user in this specific case.

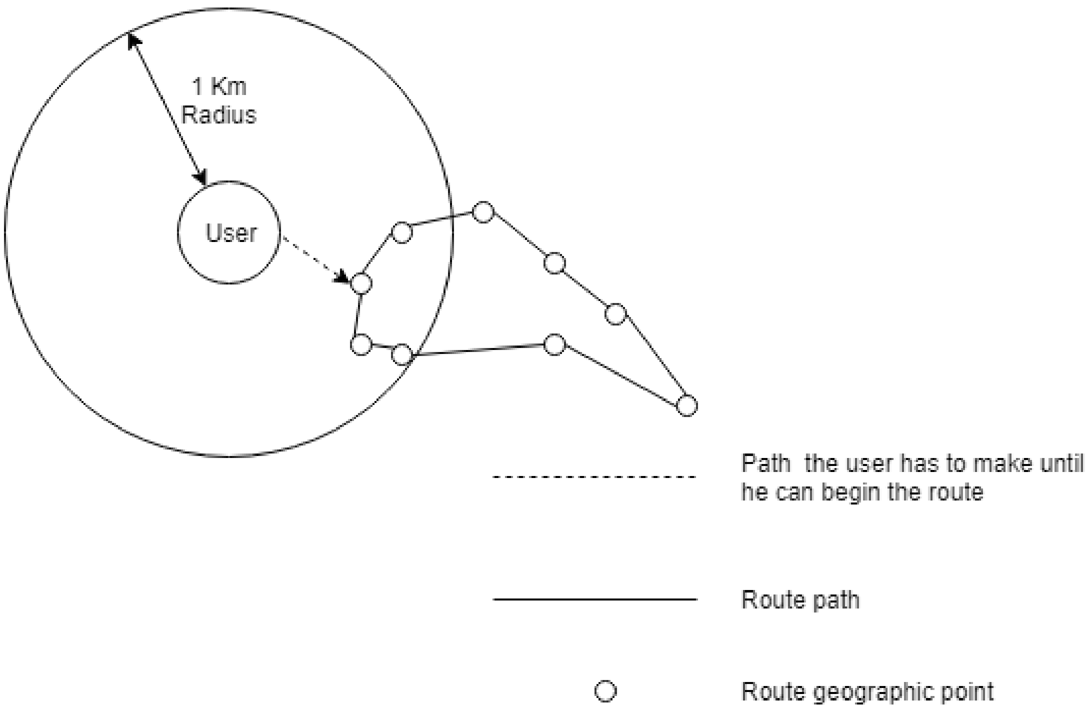

Figure 3.4: Relevant circular route example.

The third case is when a rout is not ordered and can be started either by the first or last geographic route point. An example can be seen in figure 3.5.

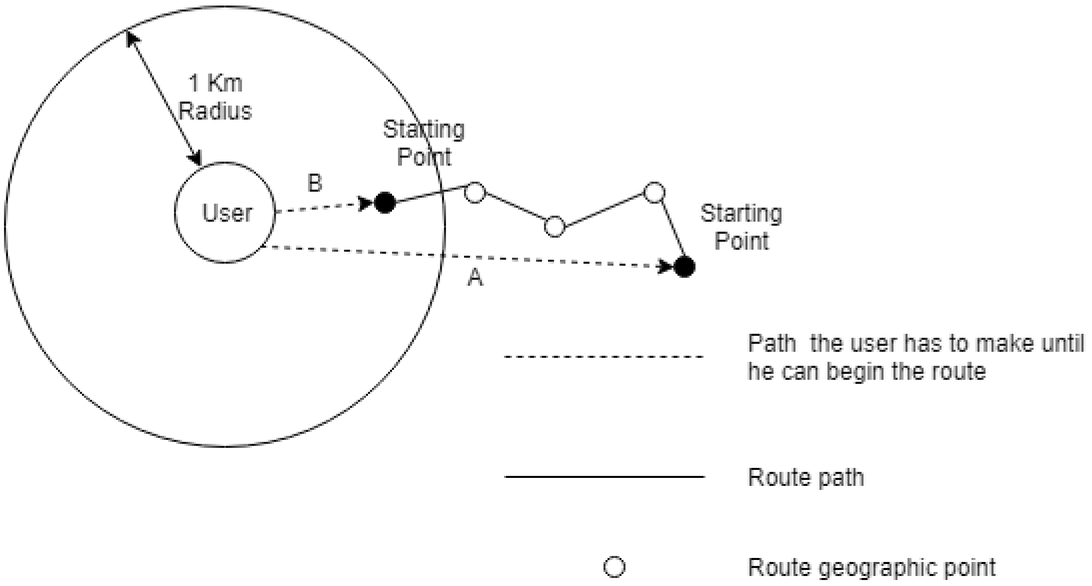

Figure 3.5: Relevant unordered route example.

Because the route does not require the user to start from the first geographic point, then the choice between the first and last is available, and in this case, as seen in figure 3.5 the path B, to the last point of the route is much shorter than path A. If the route was ordered it would not be relevant to this user, but the possibility of starting from the end makes it relevant.

After initial filters are applied the algorithm's second phase begins, being that it will vary according to the previously described fields circular and ordered. If a route is circular, the geographic location of the user (latitude and longitude) will be compared with every point each of the already filtered routes, to check if they belong to the user area. If they belong to the area they are considered relevant, if not they are discarded.

If a route is ordered the starting point of the route is the only one being compared to the user's location, if not, both the final and starting point are compared. Like the circular, the ones that belong to the area remain, the rest are discarded.

From a technical point of view, to be able to calculate if a route belongs to a user given area, the user coordinates are transformed into a Geographic Object[18], then, each of the already filtered and retrieved routes is checked for distance to that geographic object. To be noted that the construction of a geographic object is necessary because the calculations being made are not made on the cartesian plane but on the spatial one. They must be made considering a spherical object to provide accurate results (in case a long distance between route points exists). Each route contains every geographic point in its field points, which is of type JSON. Of of each of the routes, its points must be extracted and converted into the same Geographic Object type. From here, using the built in function of the PostGIS extension STDistance [57], the distance between points is retrieved, if it's within the selected radius, it's selected, otherwise it is discarded.

3.2 Social Routing API

3.2.1 Schema

The Social Routing API uses the HTTP protocol as a medium to communicate and all the data sent or received must be in the JSON format. The base endpoint of the API is : http://api.sr.

Data obtained from the API is either a single resource or a collection of resources. For example, a request made to retrieve a route will have as a response a single resource which will contain the representation of that route. If a request is made to obtain routes by location then the response will be a collection of resources containing the several route representations. This single and collection terms when associated with a resource were decisive in the choice of how much data would be returned in either of them. A request for a collection doesn't require detailed information about each element of the collection, it needs to provide general information and a form for the API user to retrieve detailed information about a specific collection element. With this idea in mind the resource representations were divided into two types, a detailed representation for when a user requests a single resource and summary representation for each element inside a requested collection, containing only the information about that element that is necessary. Examples of both a detailed and summary representation can be found in the Schema Documentation [52].

3.2.2 Authentication

The authentication has two different endpoints, one to obtain credentials to access the API and the other to refresh the already obtained credentials. To make an authenticated request to the API the user must make every request with the already obtained access token in the Authorization header of the request.

Obtaining credentials

To obtain API credentials a POST [26] request must be made to the endpoint: http://api.sr/auhtentication/google, with the id token string in the body of the request:

```
{
  idTokenString : "id token string"
}
```

The request to this endpoint receives a response containing all the necessary data to make authenticated requests to the API. An example response:

```
{
    "accessToken" : "access token 1",
    "refreshToken": "refresh token 1"
}
```

Refreshing the credentials

To refresh credentials a POST request must be made to the endpoint: http://api.sr/auhtentication/refresh containing in it's body the refresh token. If the refresh token is valid a response with a new access token and refresh token is generated.

3.2.3 Supported HTTP Methods

Due to the nature of the HTTP protocol, the API supports four different HTTP request methods [48]: GET, POST, PUT and DELETE.

GET

This method is used to retrieve resources from the API. The request:

```
GET http://api.sr/persons/1
```

retrieves a resource representing a person resource with the identifier 1. The response to this request would be:

```
{
  "identifier": 1,
  "rating": 4,
  "routesUrl": "http://api.sr/persons/1/routes"
}
```

POST

The POST HTTP method is used to create resources. It requires that the Content-Type[14]HTTP header is defined with the value application/json[10]. An example of a post request can be found in the API POST[50] documentation. A response to a POST request has an empty body and returns the location of the created resource in it's Location header. If successful the status code of a POST request response is 201.

PUT

The PUT method is used to replace or update a resource or a collection of resources. Like the post request it requires that the request contains the HTTP header Content-Type defined with application/json. The following request replaces the currently existing resource route with identifier 1 with the one sent in the body of the request. A successful response has the 200 OK status and an empty body. An example of a put request can be found in the API PUT[51] documentation.

DELETE

DELETE, as the name implies is utilized to delete a resource. The request:

```
DELETE http://api.sr/routes/1
```

deletes the route with 1 as identifier. A successful response will have the 200 OK status and an empty body.

3.2.4 Pagination

Requests that return a collection of resources will be paginated to a default value of 10 resources within the collection. A specific page can be requested with the query parameter page. If no page is specified then the first page is assumed by default. The request:

```
GET http://api.sr/persons/1/routes?page=1
```

The response would contain a collection of the first ten routes that a person with identifier 1 created. To obtain the next ten one would simply change de value of page to 2. If no further routes are available then a response with the status not found and error code 404 is returned.

3.2.5 Errors

The error responses follow the RFC standard of type problem+json[36]. An error response example:

```
{
  "type": "Social-Routing-API#unsupported-media-type",
  "title": "The requested type is not supported.",
  "status": "415",
  "detail": "The xml format is not supported."
}
```

The type field contains a link to the API documentation which further describes the type of error generated. The title describes succinctly what the error is about. The status is the corresponding HTTP status for the specific error. Detail is a more detailed message about the error encountered.

3.2.6 Hypermedia

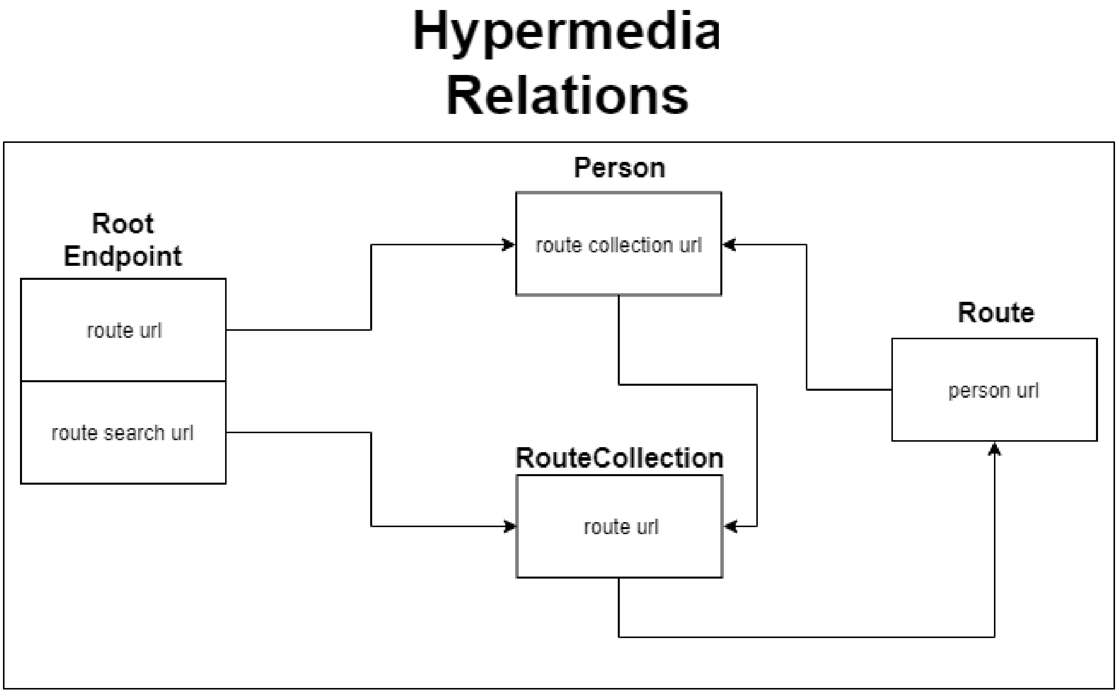

Figure 3.6: Hypermedia.

Some resources have links to other resources. Either to a parent resource or to a detailed representation of a resource within a collection. For example, a user resource haves a link to their created routes, which holds a collection of routes. That same collection haves a link to the profile of the person who created the routes. A detailed relationship diagram containing the most relevant relations can be found in figure 3.6

3.3 Server

The server uses Kotlin as a programming language and the Spring framework[56]. It's role within the system is data receival, data processing, and to respond accordingly. It is divided in three major layers, each with it's role in the Social Routing Service's system. They are the Controllers, the Services and the Repository.

The Controllers are responsible for handling the reception of an HTTP request to the service and are mapped to it's endpoints. Upon receiving a request they will use the available services to perform desired operations either over a set of received data or to generate the requested data.

The Services are responsible for processing data and communicating the the Repository layer.

The Repository layer is the only layer with direct access to the database and as such is responsible for communicating with it. The communication is made through the use of JDBI[30], a library built on top of the driver JDBC[29]. This allows less verbose code while maintaining control over SQL queries.

Besides these three major layers the server contains other important components, the Interceptor[55] and the Exception Handler[54]. There are three different implementations of the Interceptor component.

The Authentication Interceptor is responsible for user authentication before the request reaches a controller. The goal of this implementation is to avoid server overhead, resolving the authentication before the request is processed allows for a fast response if the user is incorrectly authenticated instead of continuing with the unnecessary processing of data.

The Logging Interceptor is used both before and after the request is processed to provide information regarding each request for debugging purposes.

The Media Type Interceptor is used, like the Authentication Interceptor, to avoid overhead, since if a post request is made with wrong Content-Type headers or no headers at all then the service does not support that request and can respond with an error immediately.

The Exception Handler is the component responsible for the handling of exceptions of the system. In the Spring framework there are several ways to handle exceptions, but the choice to make is to either handle the exceptions locally or globally. The handler implementation groups all the exceptions thrown by the system in a single class and produces their respective error messages. It allows for an easier work flow when treating exceptions. The global handling was chosen because most of the exceptions happen in more than one endpoint and would produce a lot of repeated code if handled locally.

As an example, the flow of a correct HTTP POST request to the routes resource that arrives on the server is the following:

- The request is intercepted by the Logging Interceptor and logs the request information.
- It is then intercepted by the Media Type Interceptor, that checks if the request data format received is supported by the service.
- The Authentication Interceptor checks the user credentials to see if the user can indeed access the service.
- The endpoint is reached in it's mapped Controller, which receives the Route information and that it maps to the correct object. In this case the Route Controller which will then call a service responsible for processing the request data.
- The service, in this case Route Service, will process the data and map it to the correct data type and make a request to the repository to store the received data.
- The repository communicates with the database, to which it sends the data in a database accepted format.
- The database stores the data and returns the identifier of the newly created Route.
- The repository receives the identifier and passes it through to the Route Service.
- The Service passes the received information to the Route Controller.
- The Route Controller builds the newly created route resource URI with the received Route identifier and maps it to the header Location of the response and returns.

3.3.1 Authentication

The nature of the project requires that user information such as user made routes is stored and because of this user authentication is required.

The service handles two main types of authentication requests, one to generate the user credentials and the other to refresh or update already existing credentials. A detailed view of the Social Routing Service and the components that interfere with the authentication process can be seen in 3.7.

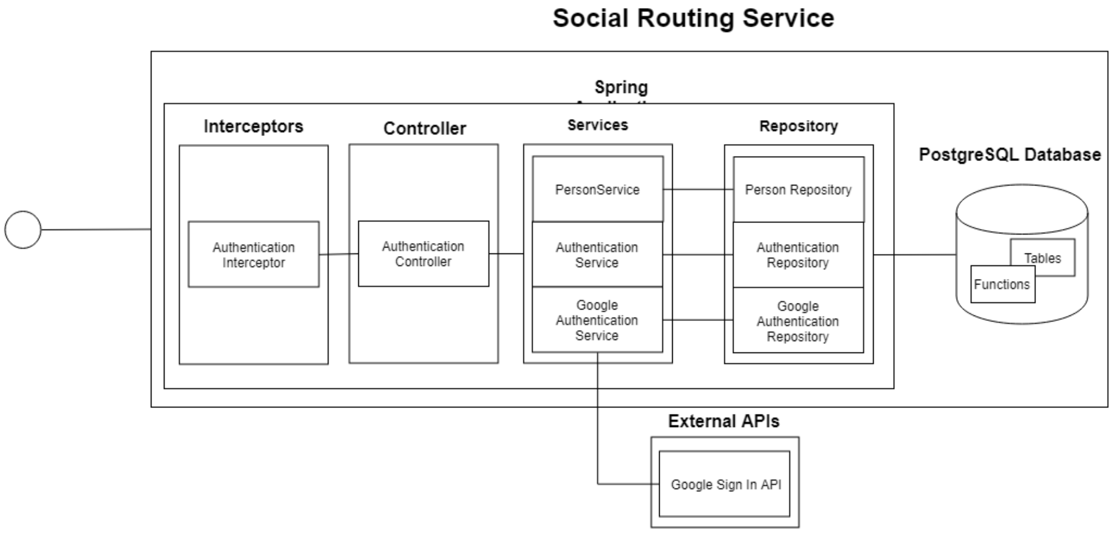

Figure 3.7: Authentication components diagram.

Generation of Credentials

When a request is received, as mentioned in 3.2.2, it must contain the id token string of the user's google account. When the controller receives it the first step is validation of the token, for that an external API is used (Google Sign In API), that has it's usage wrapped in the GoogleAuthenticationService[19]. This service is responsible for all actions pertaining Google Authentication's specific processing. Once the token is validated and the sub extracted from it, the sub is checked against the database to see if any user with it is already registered, if not, the user is created. Along with user creation, the controller requests both the access token and refresh token from the AuthenticationService's[12] available functionalities. The tokens are generated using Java's Secure Random class[47] and hashed before being stored in the database. The access token and refresh token are sent in their original form as a response to the request maker, so that further API requests have valid credentials.

Credential Refreshment

A valid API user will have both an access token and a refresh token to make authenticated requests to the API, but the tokens themselves have a duration of a day. For this reason if an API client makes a request to it and a day has passed since the token has been obtained then the client needs to update his credentials. To do that a request must be made using the refresh token, obtaining newly generated credentials from the service.

Apart from these two distinctions, the service client still also needs to worry about making authenticated requests, which can be made by adding a valid access token to the Authorization header of the response as detailed in section 3.2.2. Behind every request made to the service, the Authorization header is checked, to validate the user making the request. This is made in the AuthenticationInterceptor class [11].

4 Social Routing Client Application

The Social Routing Client Application is composed by four major components, each with it's own compromise and objective. The Activities[1] and Fragments[3] to represent the User Interface (UI) where the user can interact with the application, the View Models[8] to store and manage UI related data, a Repository that handles data operations and knows where to retrieve data from and a Remote Data Source to communicate with external components, for instance the Social Routing API, the Google Sign In API[25] or Google Maps Platform. This logic is represented in the figure 4.1.

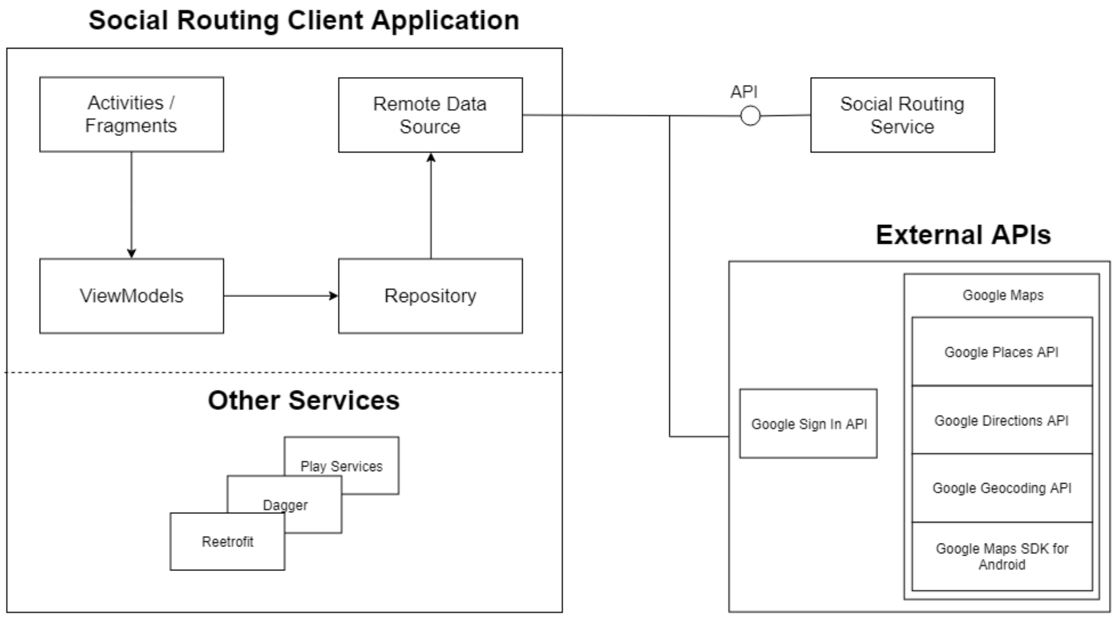

Figure 4.1: Social Routing Client architecture.

Remote Data Source

Module that has the objective of communicating with external APIs which it does by executing requests to either the Social Routing API, the Google Maps Platform or the Google Sign In API. The Component knows the structure of the HTTP request to the endpoints, like the parameters and meta-data necessary to obtain the required Response. After the request is done, the webservers[60] provide the response in the Json[31] format, however the response is deserialized using the library Jackson[28], to convert it to Object. So was defined all the input model objects, to automatically deserialize the response to object.

Repository

The Repository Handles data operations, knows where to get the data from and what API calls to make when the data is updated. A repository can be considered a mediator between different data sources, such as web services.
The application has repositories specified to the Social Routing API and another to Google Maps Platform, which contains multiple APIs. Each one as correspondent Web Service that uses the framework Retrofit[41], used to make a synchronous or asynchronous HTTP request to the remote webserver. The Repository obtains the data from the web server and can only have two possible request status: Failure or Success. It returns the data contained in a LiveData[5] because when the data updates can then be observable.
The Repository specific to our API (Social Routing API) knows all the endpoints[**endpoint TODO**] that should make the request by doing a request to the root endpoint, depending on the objective and the functionality, like the endpoints to sign in, to get user info, routes, create a route, get all categories, update a route, amongst others.
On the other hand the other repository is used to make request to the Google Webserver (Google Maps Platform) about the geocode[17] of a location, the directions to a coordinate in the map and the places of a given area.

View Models

Component used when the UI experiences a change. The View Model calls other components to load the data, and it can forward user requests to modify the data however it doesn't know about UI components, it is completely separated from them.
This component has a simple implementation, the application contains three View Models one for the Routes information (get, creation, update, search), to the User (get, update) and another to the Google Maps Platform (information of a location, places, directions). It uses the repository to obtain the data and then return it in the shape of Livedata.

Activities/Fragments

The only concern of this component is to provide a way for the user to interact with the application and with the view. All activities extend a BaseActivity that has a global behavior such as when the data is changed, is necessary to update the view and to show some view components like a toast[7], progress bar[6] or even start another activity, to be more noticeable to the user what is happening on the application.

Each Activity has its own :

- Design: defined by one or more layouts that contains buttons, images, input text, fragments (for instance the map fragment provided by Google), with which the user can interact and make actions.
- Behavior: when the data changes the view needs to be updated, behavior this, that is defined with either a success or error, using the ViewModel.

BaseActivity

- Design: No layout.
- Behavior: Is a abstract implementation of a Activity, where contains generic behavior to view components and knows how to handle changes in the Livedata from View Model, to all the Activities that implement it.

LoginActivity

- Design: Login Activity Layout.[34]
- Behavior: Where the Authentication4 happens, using the google account credentials the user can login/authenticate in the server side and a request to the Root endpoint of the Social Routing API is done.

NavigationActivity

- Design: Navigation Activity Layout.[35]
- Behavior: The client can navigate through the application functionalities with this activity. This layout contains a text box to input the location, to search for the best route possible, a list that shows the closest routes of the user current location, doing a request to the Social Routing Service with the default filters like all categories, a route with short duration and finally a panel that contains buttons to go to other activities, like RouteCreationActivity4 and UserProfileActivity4.

RouteCreationActivity

- Design: Route Creation Activity Layout.[42]
- Behavior: A core functionality of the Social Routing Client Application is in this activity, because is where the user can create a route and it's corresponding metadata. Initially is obtained asynchronously a Fragment, provided by Google Play Service[24], that represents the Google Maps[21] and right after the map is RouteDetailsActivity a form pops up asking for the desired location of the route, to be created. After the user inputs the location name, is made a request to Geocoding API[20] from google to obtain the geocoordinates to do a zoom in the map. Then request the user can click in the map to select the route points, to create the path and are saved. When the route path is finished, the user click must click in the button continue to proceed to the Places of interest, that is, is made a request to Google Places API[23] to obtain the Places Of interest near of the route points (in a range of one hundred meters) that can be added as metadata of the route. To conclude, is necessary to fill a form, in the RouteCreationMetadataActivity 4 to terminate the creation.

RouteCreationMetadataActivity

- Design: Route Creation Metadata Layout.[43]
- Behavior: The objective of this activity is the termination of the route creation. Where receives the route information that was obtained from the previous activity (RouteCreationActivity4) via Intent [4] and is needed to fill fields about the route like the name, description, categories, duration and image to fully create the route and make a POST request to the Social Routing Service to be created, if is succeeded is shown the route representation in the RouteRepresentationActivity4.

RouteDetailsActivity

- Design: Route Details Activity Layout.[44]
- Behavior: Activity to represent some metadata of the route like the name, description, categories, duration and image that are obtained via Intent and are represented to the layout.

RouteRepresentationActivity

- Design: Route Representation Activity Layout.[45]
- Behavior: Through Intent is sent the route Url[58] to this activity, to do a request to Social Routing Service and obtain all the route detailed information. The Google Maps is initialized and is shown to the user the route path with blue markers and lines that are drawn in the map and the places of interest with red markers.

RouteSearchActivity

- Design: Route Search Activity Layout.[46]
- Behavior: The Search functionality is done with specific filters (categories, duration and location) and it is done in this activity, by passing the correspondent parameters to it and is made a request to Social Routing API, if succeeded are shown in a list the results.

UserProfileActivity

- Design: User Profile Activity Layout.[59]
- Behavior: Previously was done a request to the root endpoint to obtain the Urls of the API and also comes the user profile Url of the user authenticated and is used to make a request to obtain all the information about it.

The Client Application has the minimum API level 23 and the target API level is 28, so the Platform version is the Android 9. It uses Kotlin as the only the programming language in the project which is an object oriented programming language. The goal was to improve the coding experience in a way that was practical and effectual. Kotlin is entirely compatible with Java and was specifically designed to improve existing Java models by offering solutions to API design deficiencies.
The core functionalities of the application require a map to create the routes and to show them which was done was using the Google Maps Platform. All the functionalities of the application are provided from de Social Routing Service, except all that is related to the Google Maps. All the information is obtained from the server by doing requests to the correspondent endpoint and it is always necessary to send either the token created by the server or the Google Sign-In API tokenId, used on the user registration process. The requests that are related to location retrieval or Map UI require a specific request to the Google Maps API.

Use Case

As an example, the user first experience flow of the application is the following:

- The user provides his google account credentials to authenticate with the application, which in its turn makes a request to the backend server.
- The user will be redirected to a navigation screen that contains a route search bar and a left panel menu with buttons to redirect to the screens of user profile and route creation.
- After the user searches routes using a location, a redirection to a new screen occurs in which a list of found routes is shown.
- Once a route is chosen and pressed upon, a new activity with a map is shown, where the route is represented and with a button to start Live Tracking.
- By choosing to start Live Tracking the user location is now showed as well as a path to reach the beginning of the chosen route.
- If the user wants to see the his profile he may go back until the navigation screen and click in the left panel and then in the User Profile button.
- In the user profile the user information (user rating, name, email and routes created) is shown.
- For creating a route, a button press in the bar menu is required.
- This action takes the user to a new screen that shows the map, a button to finish and a form, asking the location of the route that will be created.
- The user must then insert the location on which the map will zoom in.
- A click in the map will add the pressed location point to the route being created. If something wrong occurs the user can delete the last point of the route clicking the button on the top of the screen.
- When finished the user click in the button to fill the final form that contains the name, description and category of the route.

Internal Services

External Services

Authentication

5 Conclusion

All the functionalities are currently done but some still need work done to be improved. On the client side both the route search functionality and Live Tracking functionalities are done, albeit in a minimalistic way, the route creation and representation functionality are fully implemented and will only change upon addition of new functionalities. User authentication, using the google account to authenticate is also completed. The Server side is close to complete lacking only (complete) error handling, finish pagination and search algorithm improvement.

The expectations to this beta version are pretty similar to the current state of the system, however some problems were found configuring the entire system. 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.

Some of the final changes might be the global optimization of the system regarding all the functionalities and the core changes might be in the route Search Algorithm that should have the best possible route suggestions, the route live tracking that should send to the user some notifications to help him go through the route and reliable system tests to ensure that everything is working properly. Small adjustments to all functionalities will occur as well to guarantee that the structure and representation are in their best possible state.

Bibliography

[1] *Android Activity documentation.* URL: https://developer.android.com/guide/components/activities.html.

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

[3] *Android Fragment documentation.* URL: https://developer.android.com/guide/components/fragments.

[4] *Android Intent Documentation.* URL: https://developer.android.com/reference/android/content/Intent.

[5] *Android Live Data documentation.* URL: https://developer.android.com/topic/libraries/architecture/livedata.

[6] *Android ProgressBar documentation.* URL: https://developer.android.com/reference/android/widget/ProgressBar.

[7] *Android Toasts documentation.* URL: https://developer.android.com/reference/android/view/View.

[8] *Android View Model documentation.* URL: https://developer.android.com/topic/libraries/architecture/viewmodel.

[9] *API definition.* URL: https://en.wikipedia.org/wiki/Application_programming_interface.

[10] *application/json documentation.* URL: https://tools.ietf.org/html/rfc4627.

[11] *Authentication Interceptor implementation.* URL: https://github.com/baltasarb/social-routing/blob/master/social-routing-service/src/main/kotlin/ps/g49/socialroutingservice/interceptors/AuthenticationInterceptor.kt.

[12] *Authentication Service implementation.* URL: https://github.com/baltasarb/social-routing/blob/master/social-routing-service/src/main/kotlin/ps/g49/socialroutingservice/services/AuthenticationService.kt.

[13] *Client Application Documentation.* URL: https://github.com/baltasarb/social-routing/wiki/Client-Application.

[14] *Content Type documentation.* URL: https://tools.ietf.org/html/rfc7231#section-3.2.

[15] *DBMS definition.* URL: https://en.wikipedia.org/wiki/Database#Database_management_system. (accessed: 29.04.2019).

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

[17] *Geocode definition.* URL: https://en.wikipedia.org/wiki/Geocode.

[18] *Geographic Object Documentation.* URL: http://www.gitta.info/DiscrSpatVar/en/html/NaturOfVaria_learningObject1.html.

[19] *Google Authentication Service implementation.* URL: https://github.com/baltasarb/social-routing/blob/master/social-routing-service/src/main/kotlin/ps/g49/socialroutingservice/services/GoogleAuthenticationService.kt.

[20] *Google Geocoding API.* URL: https://developers.google.com/maps/documentation/geocoding/intro.

[21] *Google Maps documentation.* URL: https://developers.google.com/maps/documentation/.

[22] *Google Maps platform.* URL: https://cloud.google.com/maps-platform/?utm_source=google&utm_medium=cpc&utm_campaign=FY18-Q2-global-demandgen-paidsearchonnetworkhouseads-cs-maps_contactsal_saf&utm_content=text-ad-none-none-DEV_c-CRE_267331616281-ADGP_Hybrid+%7C+AW+SEM+%7C+BKWS+~+EXA_%5BM:1%5D_EMEA0t_EN_Google+Maps+Brand-KWID_43700020520504203-kwd-298247230465-userloc_1011742&utm_term=KW_google%20maps%20api-ST_google+maps+api&gclid=CLHvqYu4reMCFQ5zGwodPiEHBA.

[23] *Google Places API.* URL: https://developers.google.com/places/web-service/intro.

[24] *Google Play Services.* URL: https://developers.google.com/android/guides/overview.

[25] *Google Sign-In API Documentation.* URL: https://developers.google.com/identity/sign-in/web/sign-in.

[26] *HTTP POST request documentation.* URL: https://tools.ietf.org/html/rfc2616#section-9.5.

[27] *HTTPS RFC.* URL: https://tools.ietf.org/html/rfc2660.

[28] *Jackson documentation.* URL: https://github.com/FasterXML/jackson.

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

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

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

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

[33] *Kotlin Documentation.* URL: https://kotlinlang.org/docs/reference.

[34] *Login Activity Layout documentation.* URL: https://en.wikipedia.org/wiki/Geocode.

[35] *Navigation Activity Layout documentation.* URL: https://en.wikipedia.org/wiki/Geocode.

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

[37] *PostGIS Extension.* URL: http://postgis.net/.

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

[39] *PostGreSQL Functions Documentation.* URL: https://www.postgresql.org/docs/9.1/sql-createfunction.html.

[40] *PostGreSQL Queries Documentation.* URL: https://www.postgresql.org/docs/9.4/queries-overview.html.

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

[42] *Route Creation Activity Layout documentation.* URL: https://en.wikipedia.org/wiki/Geocode.

[43] *Route Creation Metadata Layout.* URL: https://developers.google.com/places/web-service/intro.

[44] *Route Details Activity Layout documentation.* URL: https://developer.android.com/reference/android/content/Intent.

[45] *Route Representation Activity Layout documentation.* URL: https://developer.android.com/reference/android/content/Intent.

[46] *Route Search Activity Layout documentation.* URL: https://developer.android.com/reference/android/content/Intent.

[47] *Secure Random Documentation.* URL: https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html.

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

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

[50] *Social Routing API POST documentation.* URL: https://github.com/baltasarb/social-routing/wiki/Social-Routing-API#post-example.

[51] *Social Routing API PUT documentation.* URL: https://github.com/baltasarb/social-routing/wiki/Social-Routing-API#put-example.

[52] *Social Routing API Schema Documentation.* URL: https://github.com/baltasarb/social-routing/wiki/Social-Routing-API#schema.

[53] *Social Routing Service's DBMS documentation.* URL: https://github.com/baltasarb/social-routing/wiki/database-management-system.

[54] *Spring Exception handling documentation.* URL: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/mvc/method/annotation/ResponseEntityExceptionHandler.html.

[55] *Spring interceptor documentation.* URL: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/HandlerInterceptor.html.

[56] *Spring MVC Documentation.* URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html.

[57] *STDistance function documentation.* URL: https://postgis.net/docs/ST_Distance.html.

[58] *URL definition.* URL: https://pt.wikipedia.org/wiki/URL.

[59] *User Profile Activity Layout documentation.* URL: https://developer.android.com/reference/android/content/Intent.

[60] *Web Server definition.* URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server.