FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Mobile application for Prague public transport visualization
Student:	Galymzhan Dosmagambet
Supervisor:	Ing. Marek Sušický
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of winter semester 2021/2022

Instructions

The main goal of this thesis is to design, implement, and test a mobile application for public transport visualization. The system should enhance the work of Ing. Jan Spolek, "Vizualizace a predikce pražské příměstské dopravy" and should use the same data structures. The application should not only visualize the current public transport but provide statistics and offer some delay prediction.
The mobile application should implement a user account via which the users could customize the provided information and bus lines. The student should do the analysis and define these customizations (for example favourite bus lines). The application can offer to store some historical data but the user should give consent to do that. Consent can be every time changed and data erased. Mobile application should be responsive and for Android phones only.

Electronically approved by Ing. Jaroslav Kuchař, Ph.D. on 28 August 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Mobile application for Prague public transport visualization

Galymzhan Dosmagambet

Department of Software Engineering
Supervisor: Marek Sušický

February 11, 2022

Acknowledgements

I would like to thank my family and friends for the moral support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 11, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Dosmagambet, Galymzhan. *Mobile application for Prague public transport visualization.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Role veřejné dopravy je v současné době zásadní. Je přítomen v každém moderním městě a je možné jej maximálně využít pro socioekonomický rozvoj. Proto si tato bakalářská práce klade za cíl vytvořit mobilní aplikaci OS Android a vhodnou infrastrukturu pro vizualizaci MHD města Prahy. Navrhované řešení poskytne kompletní systém, který se skládá z mobilní aplikace, webového serveru a databáze. Systém nabídne jiný přístup k mobilním aplikacím MHD. Kromě vizualizace a sledování v reálném čase poskytne funkce personalizace, jako je vytvoření účtu, označení oblíbených tras a statistiky zpoždění. Výsledkem je vývoj prototypu systému, který slouží jako důkaz koncepce.

Klíčová slova Mobilní aplikace, veřejná doprava, OS Android, vizualizace, Golemio

Abstract

The role of public transport is fundamental in the current era. It is present in every modern city, and it is possible to take maximum advantage of it for socio-economic development. Therefore, this bachelor's thesis aims to create an Android OS mobile application and appropriate infrastructure for visualization of public transport of the city of Prague. The suggested solution will provide a complete system which consists of a mobile application, a web server, and a database. The system will offer a different approach to public transport mobile applications. Apart from visualization and real-time tracking, it will provide a personalization functionality such as account creation, marking favorite routes and statistics of delays. As a result, a prototype system is developed and serves as a proof of concept.

Keywords Mobile application, public transport, Android OS, vizualization, Golemio

Contents

List of Figures

List of Tables

Introduction

Public transport is essential for modern society and comes with meaningful benefits. It has a favorable effect on the environment by leaving a smaller carbon footprint and reducing the use of private transport, which causes most greenhouse emissions. Furthermore, it serves people to advance their lifestyle and our community would not work properly without it and highly depends on it. This factor boosts growth in a variety of other spheres and increases the standard of living. To attract and encourage the population to adopt more to urban transportation services rather than driving private vehicles, the mobile application for public transport was developed.

Mobile applications cooperate to make full usage of public transport. They increase the convenience of use for people and provide services affecting their day-to-day life. Mobile applications for public transport helps to save time for journey and simplify planning, by providing schedules, shortest routes and other useful services. Nevertheless, it also has a good effect on the environment. Most greenhouse emissions are caused by private cars and the attraction of people to public transport via mobile applications improves this effect.

There are several solutions with different perspectives to the problem. They cover most of the basic needs for tracking in visualizing public transport. Standard functionality is finding the shortest way between 2 addresses or stops and listing time schedule for routes. Some of them like PID Litacka[1] also offer purchase of tickets, ticket pass management, places search and more.

However, these applications mostly do not support features for personalization and proper visualization, due to different approaches to facilitate the use of public transport. For example, real-time tracking of vehicles, as well as providing delays, improves the visualization, therefore allowing passengers to plan their journeys more precisely and adjust to the current transport situation faster. Another advantageous information is historical data such as delays for the past week or average delay. There are ways to improve personalization too. In general, people use public transport on daily basis and know destinations and routes they need to take. Thus one of the positive assets

would be marking favorite routes or trips.

The main goal of this thesis is to develop a mobile application for visualization of public transport of the city of Prague with personalization functionality. This application should try to fulfill lacking features and take into account the shortcomings of existing alternatives.

Aims

The main aim of the bachelor thesis is to extend existing master's thesis[2] by designing and implementing a mobile application for Android OS and appropriate infrastructure. Infrastructure should include Web a server and a database. The application suppose to visualize public transport of Prague on the map and present additional information about it. User should be able to create an account to mark their favourite routes and trips. In summary, several subgoals can be extracted, to better describe what needs to be done:

- Develop a mobile application for Android OS.
- Develop a Web server with REST API support.
- Use, extend and adapt existing database inherited from master's thesis[2].
- Provide visualization of public transport.
- Support log in and registration functionality to create user accounts.
- Provide functionality to allow users to mark favourite routes and trips.
- Provide delays and delay statistics for the last 7 days for favourite trips.

Chapter 1

Analysis and design

1.1 Related work

Arrangement and support of public transport is the non-trivial problem of any city and it is hard to find the best resolution, thus most of the time travelers have to face compromises. Routes can be changed or canceled, time schedule may differ for every day. This brings inconvenience for people and makes public transport harder to use without any guidance. Mobile applications try to help the community to provide services to use public transport advantageously, and different solutions solve different issues. These mobile applications can be divided into 2 major groups by common features. The first group consists of apps provided by tech giants like Google, Microsoft, Yandex, etc. They do not focus specifically on providing a day-to-day solution, but rather follow a general approach to apply it everywhere. The main goal of the second group of apps is to come up with a localized solution that will be specific to the particular city, hence in this paper, only Prague-related mobile applications will be reviewed.

1.1.1 Global solutions

Google maps is one of the most popular map services that exist. However, it was designed to be generalized and support a variety of features simultaneously. In this service, users can search for almost any public place, like a museum, theatre, school, monuments, etc. Users can leave reviews about any presented places and the administration of the places can provide photos, schedule, website and other useful information. Google maps is also able to find a route to a given destination with different travel options: public transport, car, bicycle or by foot. Although information about public transport is not reliable enough to use it on daily basis. Because it is very ambitious to keep up-to-date such information along with everything else Google maps maintain. Moreover, this is a global service, therefore it has to track not one

but thousands of cities, which decreases reliability even more. Google maps is a utility to search public places and obtain their location or address.

1.1.2 Local solutions

Pubtran[3] - is a mobile application for Prague public transport. Its main feature is to find the shortest way between 2 locations and provide a number of possible connections, with the opportunity to add an in-between stop. It also allows user to choose the type of public transport: for example, user can filter out results with train or tram, connections with wheelchair accessibility, or do not apply any filter. Another convenient feature is to display path on map including transport connection. There is another option for filtering in the main menu, where user can choose a time of departure or arrival. Additionally, Pubtran allows you to purchase transport tickets, but the price can be by 1-2 czk higher. Prime disadvantage is the inability to show the current location of the vehicle or present any historical data of a given or favorite route or trip.

Czech Public Transport IDOS[4] - is another mobile application for Prague public transport. This application implements a similar set of conveniences. Although it extends this set with a more advanced search option. User can choose transport operators and minimal interchange time. Aside from the disadvantages of Pubtran, IDOS is not a completely free application. It contains ads, and in order to remove them, the user has to pay. Another additional downside is data collecting. This application collects data about users in favor of selling it or showing specific ads.

PID Litacka[1] - is one more mobile application for Prague public transport which is integrated into PID Litacka regional transport system. This application has practically all features and search options as described above but drastically differentiates from them. It mainly focuses on the PID litacka system hence on managing Litacka card and purchasing tickets. Unlike in other apps, the user buys tickets and monthly passes within the PID Litacka system and not by SMS. This allows track transaction history which is linked to the user account and can be displayed not only in the app. This is a useful feature for users PID Litacka card especially because it is used as the main system for transport passes. In some cases, PID Litacka application can provide enough information for public transport controllers and can be used instead of a physical card.It has some additional positive sides like presenting plans for all kinds of public transport and further useful information for passengers.

In summary, all these application address specific problems, which does not cover the whole domain. This thesis offers a solution in form of a mobile application which will provide visualization and different personalization user experience.

1.2 Requirements analysis

In order to create a mobile application, it requires a server, which will take responsibility for providing the data. The server should have most of the business logic of the system and act as a core. The server will be connected to the database, inherited from master's thesis[2] and provided by supervisor Marek Sušický. Therefore, the system is required to have three main components: mobile application, web server and database. The next two subsections will be discussed Functional and Non-functional requirements for the system. It will describe exact functionality and expected behavior.

1.2.1 Functional requirements

- FR1: Account. User should be able to create an account for personalization purposes.
- FR2: Error handling. If application will encounter any error it should properly handle it and not crash. The same should be applied to the server.
- FR3: Delays. Application should provide delay for given vehicle.
- FR4: Save favourite Routes/Trips. User should be able add Routes/Trips in "favourites".
- FR5: Search Routes. User should be able to search among Routes. Search does not require account or any sort of logging in.
- FR6: Provide historical data for favourite Trips. Application should provide historical data for the last week for given favourite trip. Historical data consists of delay time for each day.
- FR7: Show live-time position of the vehicle. Application should display live time position of chosen vehicle.

1.2.2 Non-functional requirements

- NFR1: Scalability. Mobile application will use specific REST API, so the server can be changed easily.
- NFR2: Compatibility. Android releases new OS API every year, therefore not all android smartphones are capable of running newest applications. Thus this application should support an API, which will cover 70% of users.
- NFR3: Security. The system should has authentication and authorization functionality in order to protect Users data.
- NFR4: Usability. The UI should be intuitive.

1.3 Data source

The Golemio data platform[5] is operated by Operátor ICT[6], a.s., which is established and wholly owned by the capital city of Prague. Its services are available to the municipality, 57 city districts, city companies, contributory organizations and other organizations established or owned by the capital city of Prague, and in the form of selected open data to the public. DP Golemio services (integration, storage, data visualization, etc.), as well as regular team services of the Golemio Data Platform (technical consultations, a consultancy with the preparation of tender documentation, etc.), are available within the tariff defined by the contract for the municipality and city districts.

The platform is designed for different clients such as municipality and city districts, city companies, commercial entities and the general public. The goal is to arrange suitable data for analysis and reports for these clients. The general results of the monitoring should lead to improvements in public and commercial services. For different clients, The Golemio team can offer diverse services and options that they can agree on. However, for the public, it is more straightforward. The Golemio platform has an open public REST API. This is an expedient reason for using in the original thesis on which this project is based. Data are consumed through the API and stored in the database to be used and visualized in the mobile application. The Golemio's public API provides a variety of contrasting types of data: public transport, parking, air quality, public space, traffic, waste, pedestrians, traffic restrictions. This thesis will use only data of public transport.

1.3.1 Public API

The Golemio's API consists of several end-points which provide GTFS (General Transit Feed Specification) data about the city's public transportation schedules. This public API shall be used to fill with data the system's database.

GTFS Services[7] displays on which days it is online for given service.

GTFS Routes[8] displays data about given route. Route can be used to find general information and use it to get more specific through trips.

GTFS Shapes[9] displays path of the specific trip. It is a number of geo-points which construct the path on the map.

GTFS Stops[10] returns general information about given stop including coordinates, name, parent station ID.

RealTime Vehicle Positions[11] describes data of given vehicle and expands it. It has data not only about specific vehicle, but also information about previous stop, next stop and other.

GTFS Trips [12] displays data about given trip. Trips are real instances of the routes. Specific trip holds the most important data for visualization for the user. Trips are connecting link between all other data. They have

references to route, shape, service. Individual trip also has information about itself: trip head-sign, wheelchair accessibility, direction and other.

1.4 Design analysis

The system has to be easy to maintain and expand. This will allow to add new features easier and faster, develop new components and expand functionality. In order to make a scalable system, it has to be split into different logical layers which will have different responsibilities. This will give an opportunity to substitute them with different implementation if necessary.The whole portal will consist of 3 main components: Database, Backend server (PID-Backend) and Graphical User Interface (GUI) i.e. Android Mobile application (PID-MobApp). Therefore components will not be dependent on each other but on the Application Programming Interface (API). Namely, if any component is replaced, it has to only support an API, but implementation can be completely different. For example, in the future, PID-Portal might require to support IOS. In this case, will be necessary to develop only the IOS application which will support Backends API. As a choice Database can be replaced with a database for another city. Overall this will bring colossal flexibility. The architecture design of each component will be discussed Further.

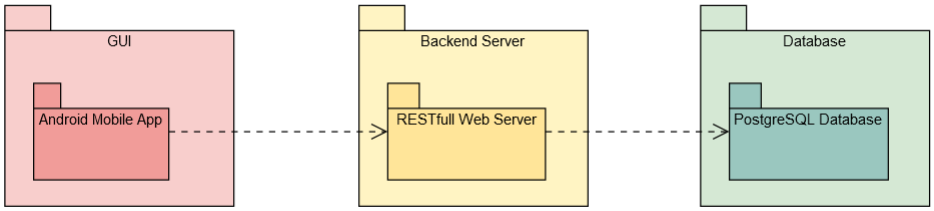

Figure 1.1: General Architecture Diagram

1.4.1 UI Design

In order to understand how PID-MobApp will look like, it is required to design a draft or wireframes. On these wireframes schematically will be shown main features and states of the application.

On the main screen will be displayed map with stops, ”Search” bar and ”Hamburger” icon representing menu button as demonstrated on the figure number 1.2 (a). If user will tap on any stop, application should display routes that go through chosen stop and when next vehicle for every route will arrive as shown on the figure number 1.2 (b). User will be able to select one of the offered routes. This will activate another view with trip details. In this case

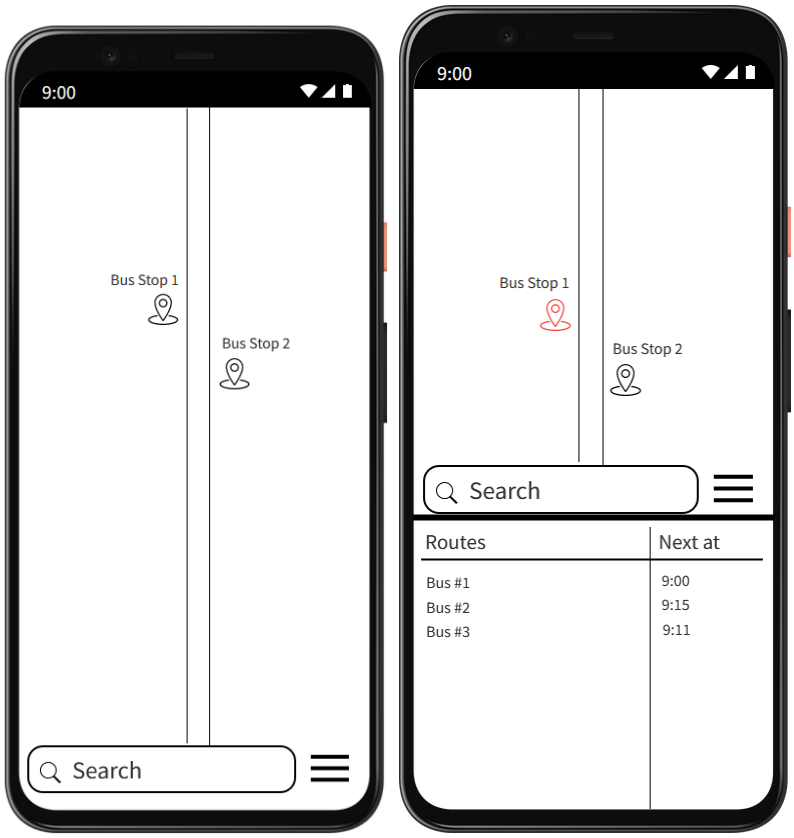

Figure 1.2: (a) Main screen (b) Stop view

details of the closest trip of the selected route to the selected stop will be displayed. At the same time track of the trip will be drawn on the map along with icon representing location of the vehicle. This is demonstrated on the figure 1.3 (a). In the trip details will presented information such as direction, head sign, if it is wheelchair accessible and if bikes are allowed.

User will be able to search for some specific route via Search bar. By tapping on it Search view will be activated as shown on the figure number 1.3 (b). In the input view user will type route number and after results will appear below. Then user can select route from the results. Then Search view will be removed and Route view will appear at the bottom with Route details. Route details will contain information as starting stop, terminal stop, short name and long name. Likewise in Trip view, track line will appear with multiple point icons representing vehicles locations following the route. User will be able to tap on one the icons to see trip details dedicated to the selected vehicle. The same Trip view will be displayed as in case of opening it via stop.

The menu button will open a side bar with login option if user is not logged in. User can login or register via Login or Register views like shown

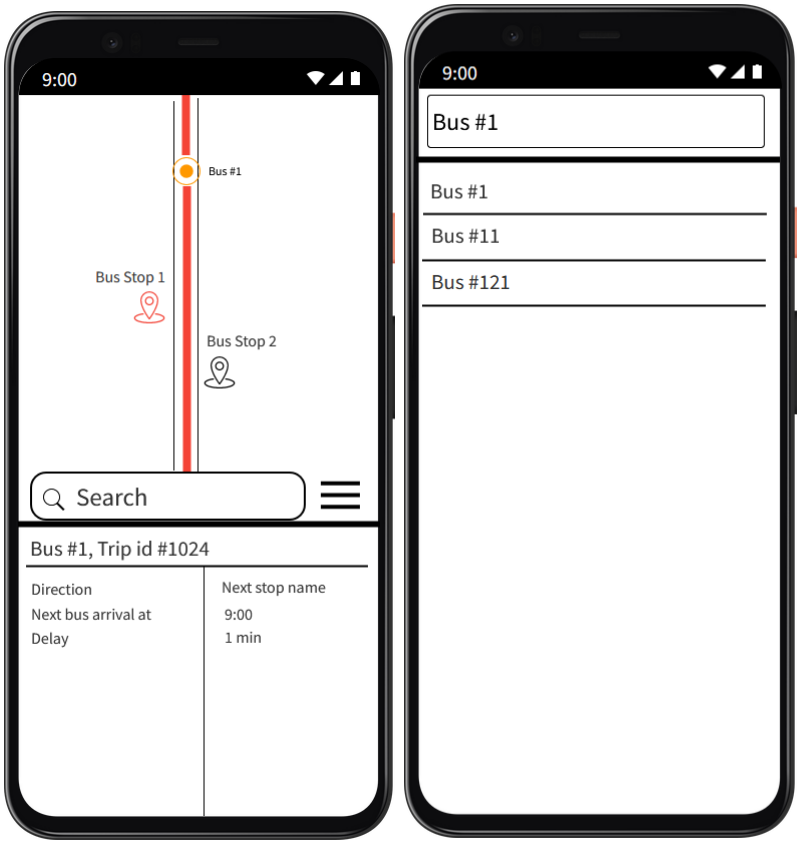

Figure 1.3: (a) Trip view (b) Search view

on figure number 1.4. Logged in user can mark and unmark favourite trips and routes on the Route and Trip views. ”Heart” icon will crop up on the right-hand side next to the name of a trip or route on the Trip view or Route view as demonstrated on the figure 1.5 (a), which can be tapped to mark. Additionally, on the menu will appear supplemental options to see favourite routes or trips. They will be listed on the corresponding views: Favourite Trips and Favourite Routes. User will be able to select favourite trip or route and appropriate view will appear on the screen.

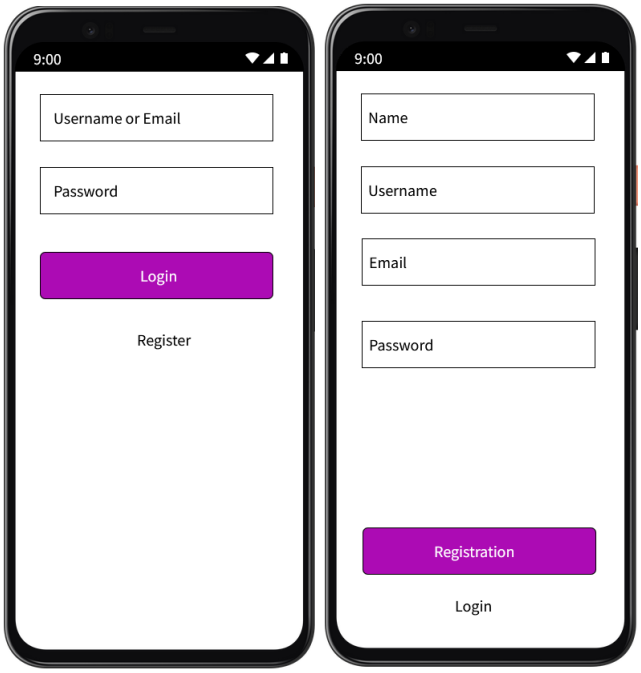

Figure 1.4: (a) Login (b) Register

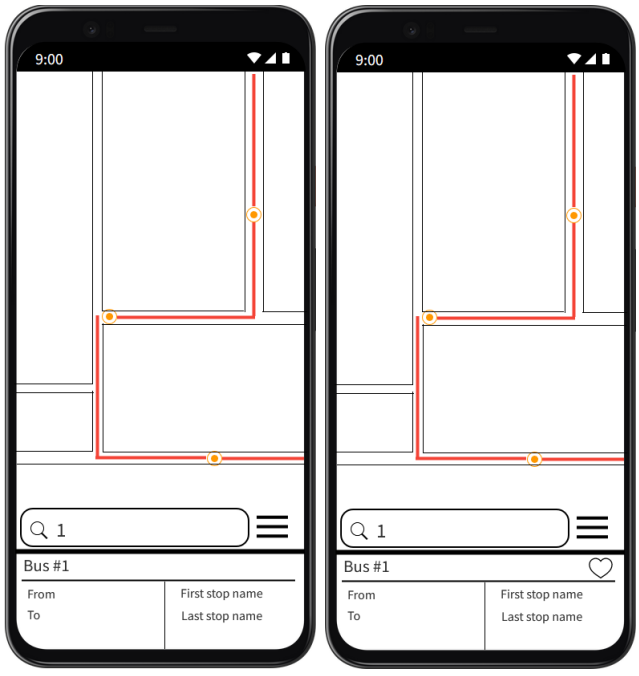

Figure 1.5: (a) Route view (b) Route view when logged in

1.4.2 Database

The database should be filled and updated by external application or manually. This is due to frequent changes in Golemio's public API. If the mobile application would use it directly, there might be a potential failures and errors, because application will not know about new API. Moreover, in case of API changes, new version of the application has to be released, which might take some time. This is a threat for comfortable usage. Therefore it is necessary for PID-Portal has its own server and database. And fill the database from outside, for instance by another component or application, which will be responsible for supporting Golemio's API and filling up the database.

Original Database

The original database was inherited from master's project[2] and provided as a part on top of which PID-Portal has to be build.

The tables in the database represent data provided by Golemio's API and have similar information and properties as shown on the figure number 1.6. The database is build around main entity - Trip. Trip is and abstract entity of a vehicle following specific route.

Shapes table is a representation of sequence of geo points. This sequence will be used to draw a track or path on map.

Services table describes availability of a trip. This can be used to display only relevant trips.

Stops table represents a stop and basic information about it, like name and coordinates. That information will be used to display stops on map.

Positions table contains more specific information about trip's position: location, next stop and previous stop, traveled distance and other. This table contains live data and intended to be updated very frequently. it is necessary to track vehicle in real time.

Vehicles table has details about specific vehicle assigned to a specific trip. This information further describes trips. This table should be kept up to date like *Positions* table and for teh same reasons.

Routes table defines routes and holds main general information about them.

Trips table is a connecting link between all information in this database. It contains mostly of foreign keys, thus all other information can be found through that table.

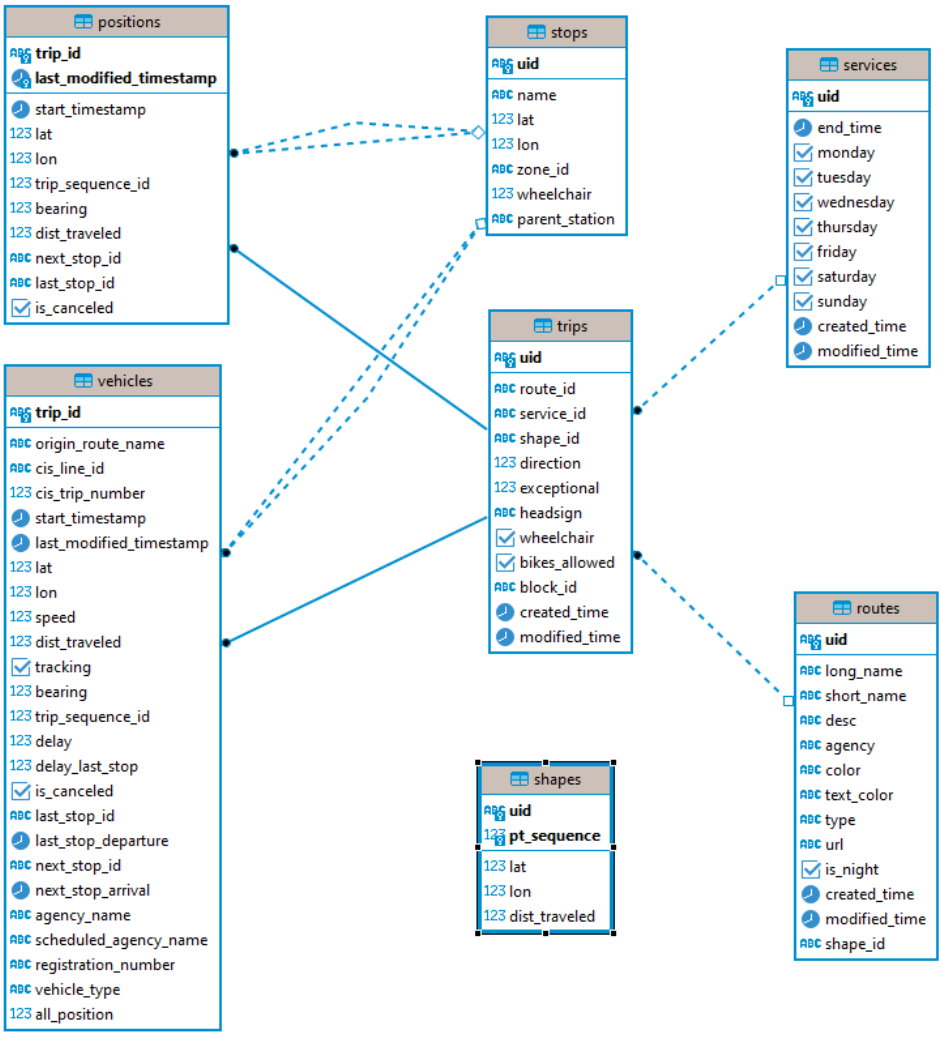

Figure 1.6: Original Database Diagram

Extended Database

The original database does not contain all the mandatory data for the PID-Portal to work properly. Therefore extra tables and fields were added, but initial structure was not changed as shown on the figure 1.7.

New tables are: Users, Users_Favourite_Trips, Users_Favourite_Routes, Trip_Stops and Delays. Table *Users* is required to store information about users and their preferences. Attributes:

- id: ID of user.
- username: username of user.

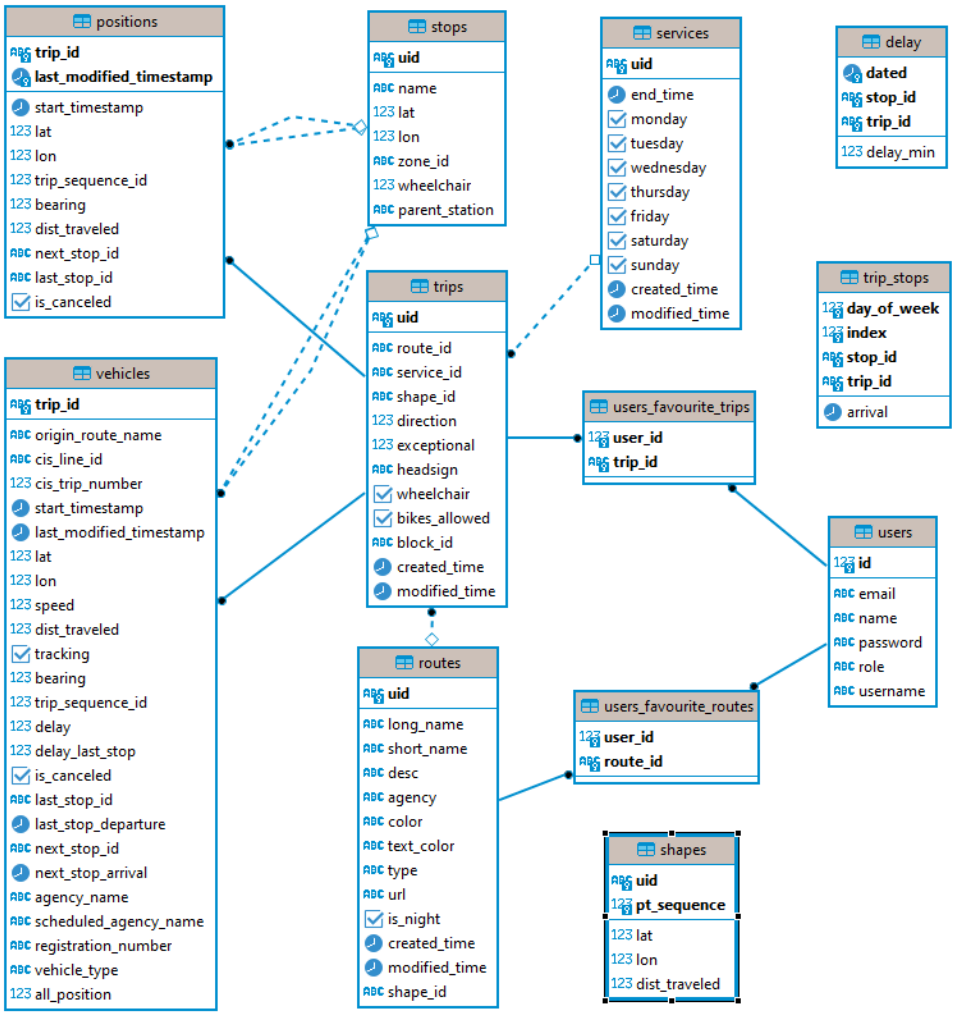

Figure 1.7: Extended Database Diagram

- password: user's password.
- email: user's email.
- name: user's name.
- role: user's role. At this moment it is not required for more types of roles, but it will be unavoidable in the future work. Thus it is better to take role into account at this moment of the development

Tables *Users_Favourite_Trips* and *Users_Favourite_Routes* are conceptually the same and their purpose if to store favourites routes or trips. Attributes:

- user_id: ID of user.
- favourite_routes_uid / favourite_trips_uid: favourite routes / trips.

Table *Trip_Stops* represents ordered list of stops visited within specific trip. It will be used to see and follow schedule of individual trip. That means trip will have numbered list of stops with expected arrival time. Attributes:

- trip_id: ID of a trip.
- stop_id: ID of stop visited within given trip.
- index: index number of the stop.
- day_of_week the day of the week. It can be week day, weekend or exceptional.
- arrival: time of expected arrival to a given stop.

Table *Delays* will contain delays of specific trips, that are marked as favourite by any user. This table will be updated by the PID-Backend if any favourite trip exists. Attributes:

- ddate: date of delay
- trip_id: ID of a trip.
- stop_id: ID of stop.
- delay_min: actual delay in minutes.

1.4.3 PID-MobApp Architecture

The architecture of PID-MobApp follows same design principals - flexibility and maintainability as the whole PID-Portal, within the constraints of Android mobile app development. PID-MobApp architecture will subsist of 3 layers: Presentation, Business, Persistence. And will follow View-ViewModel-Model architecture pattern, respectively to the layers.

Presentation layer will contain Android UI elements or rather Views, called activities or fragments. View's main responsibility is to present and display data to a user and consume user input events. After user's interactions, inputs and actions will be processed in Business layer by ViewModel.

PID-MobApp will have a several Activities which will contain inside Fragments. It gives an opportunity to reuse fragments in different activities.

Login activity will handle login and registration inputs. It will contain Login fragment and Registration fragment and follow the design on figure number 1.4.

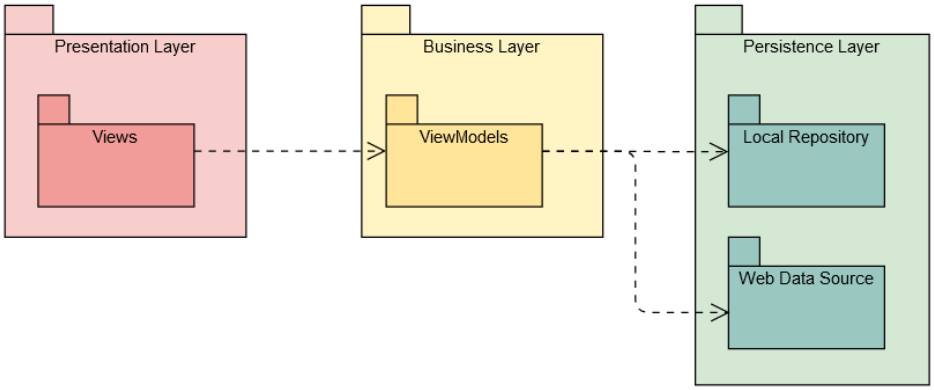

Figure 1.8: PID-MobApp Architecture Diagram

Maps activity is a main activity. It will handle main functionality: display map and map interactions, search of routes and display results, draw route line and provide information about chosen route. Therefore Maps activity will contain following Fragments:

- Search Fragment will be placed at the bottom when inactive. It will an input view with ”Hamburger” icon. Tapping on the icon will open up Navigation Menu. Tapping on Search input view activates Search Fragment and displays it on full screen.
- Search results Fragment will be a part of Search Fragment. Here search results will be displayed in a scrollable view. By clicking on any result, Route fragment for chosen route should be opened.
- Stop fragment will present an information about chosen stop along with scrollable view with list of routes going through that stop with an arrival time. By clicking on of the routes Trip fragment for closest trip to the stop of this route should be opened.
- Route fragment will show the name of the route and its direction from the starting stop to the final.
- Trip view will contain an information about a trip including list of delays if it is marked as favourite and user is logged in. AS well it will contain the average time of delays.
- Navigation Menu will have 1 option - Authorization, when user is not logged in and three when logged in: Favourite Routes, Favourite trips and logout. Favourite Routes and Trips buttons lead to corresponding

scrollable views with lists of routes and trips: Favourite Routes Fragment and Favourite Trips Fragment.

The application will have a *Database*. The database will contain user information and users favourite trips and routes.

1.4.4 PID-Backend Architecture

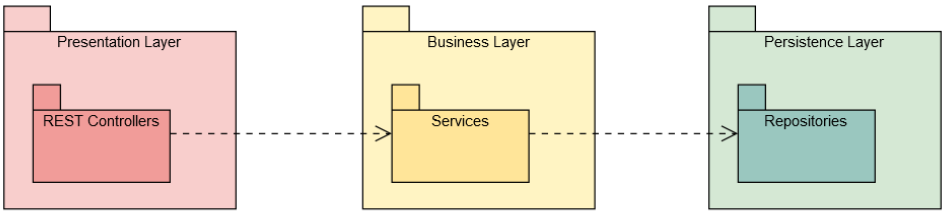

Figure 1.9: PID-Backend Architecture Diagram

Architecture of the Backend likewise architecture of the whole PID-portal is obligatory to be flexible and modular enough because of the same reasons, mentioned above. The approach is analogous to PID-MobApp architecture. Therefore the structure will be divided into 3 layers: Presentation, Business, Persistence. This is displayed on the figure number 1.9. Each layer has a specific role and responsibility within the architecture. Layers are isolated from each other, having no knowledge of the inner workings of other layers. Each layer communicates only with the layer beneath it. They will have marginally different logic behind, however abstractly have same purpose.

Presentation layer consists of REST API controllers. These controllers are triggered or called by the client-side or GUI in case of PID-Portal. The mobile application will be calling this REST API in order to obtain and present requisite data to a user. The Presentation layer's responsibility is to process http request, gain response data from Business layer and send a suitable response to the client-side. The Presentation layer must communicate only with Business layer and must not have any business logic. This characteristics will give an opportunity to add new API. Simultaneously separation of REST API controllers from business logic will make them independent of business logic's implementation. Additionally any other client application like web application, IOS application and others can use the API of Presentation layer and consequently full functionality of Backend. In order to fulfill PID-MobApp needs the Presentation layer should have following REST API like presented on the table 1.1.

Business layer contains business logic and services responsible for its implementation. There are several Service classes for each logical entity like

Route Service, Trip Service, User service, etc. Hence the design remains adjustable and Service's implementation can be easily replaced. Although Business layer should not have any direct interaction with persistent data. It must never has a direct connection do a Database or any source of data. Instead, it should use Persistence layer. The main logic in this layer focuses on preparing data for GUI requests and handling account interactions, like sign in/sign up features and marking favourite routes and trips. In addition, here will be a process responsible for updating Delays table. It will be run periodically checking *delay_last_stop* attribute in Vehicles table and updating records in Delays tables. More details about delays management are in the section 2.1.

Persistence layer is an adaptor between Business layer and persistent data or Database. It is crucial to include such layer in case of any Database changes. In case of changes of the database schema only modification of Persistence layer will be required. Therefore other layers will still be valid and process requests correctly without additional changes.

Another part of the PID-Backend is Model. Model represents data that the application will be working with. Each Model class will be corresponding to the entities in the database. This classes will be mapped onto database tables using an Object-Relational Mapping tool (ORM). Used technologies are described in the section 1.5

Name	Base URL	Additional path	[METHOD] Description
Authentication	/api/auth	/signup /signin	[POST] Sign up. [POST] Sign in.
Route	/api/route	/id /id/trips /search/name /id/trips_vehicles /id/vehicles	[GET] Route by ID [GET] Trips for Route [GET] Search Route by name [GET] Trips and Vehicles for Route. [GET] Vehicles with locations.
Shape	/api/shape	/id	[GET] Shape by ID
Stop	/api/stop	/ /id /name /id/routes /id/routesTime /stopId/routeId	[GET] All Stops [GET] Stop By ID. [GET] Stop by name. [GET] Routes that going through Stop. [GET] Routes with next time arrival. [GET] Closest Trip coming by Stop and Route ID.
Trip	/api/trip	/id /tripId/stopId	[GET] Trip by ID [GET] Trip By ID with delays to given Stop.
User	/api/user	/routes /routes/routeId /routes/routeId /trips /trips/tripsId /trips/tripsId	[GET] favourite Routes [POST] Add new favourite Route. [DELETE] Delete favourite Route. [GET] Favourite Trips. [POST] Add new favourite Trip. [DELETE] Delete favourite Trip.
Vehicle	/api/vehicle	/id /route/routeId	[GET] Vehicle by ID [GET] All Vehicles of given Route.

Table 1.1: Table of REST API endpoints.

1.5 Technologies

In this section are described used technologies.

1.5.1 PID-MobApp

- *Android SDK* [13] is a Android Software Development Kit that includes a comprehensive set of development tools. its version corresponds to a version of Android OS. For this project was chosen Android SDK 28, because it covers approximately 70% of all devices on Android OS.
- *Kotlin*[14] is a programming language and was chosen as main language for PID-MobApp. But this still leaves an option of using Java because of their compatibility.
- *Google Maps platform*[15] one of the most popular maps frameworks. This framework was chosen because it has huge support for Android OS, has documentation for Kotlin, has multiple platform support, has multiple cities support, has
- *Retrofit*[16] is an HTTP client that is used to communicate with PID-backend
- *Room*[17] was used for local database. It will store user information including favourite trips and routes.

1.5.2 PID-Backend

- Java[18] is a programming language and computing platform that was chosen as a main language for the PID-Backend. Because it is enterprise-oriented, it is supported by a huge community and there are many frameworks that helps developing any size applications.
- Maven[19] is a build automation tool used primarily for Java projects. It will be used for building the application.
- Hibernate[20] is an object–relational mapping tool for the Java programming language. it will be used
- Springboot[21] is a framework for java. It allows to create web server with minimum configuration.
- Spring security [22] is an authentication and access-control framework. It is an addition to Springboot framework which will e used for authorisation and authentication functionality.
- Log4j[23] is a Java-based logging utility that will be used to log certain events for better maintainability.

- JJWT[24] is a library that supports JWT. It allows to generate and manage JWT.
- Lombok[25] is a java library tool that is used to minimize/remove the boilerplate code and save the precious time of developers during development by just using some annotations.

Combination of all these frameworks provides enough of functional power for the development.

Chapter 2

Implementation

The implementation was going accordingly to the design described in the previous part without deviations and the final result meets the requirements established in the section 1.2. However, there were several issues during the development process. In the section 2.1 I described problems faced and the solution applied. In the section 2.2 are presented results of the implementation part.

2.1 Problems and solutions

Database I was provided with a database Docker image and Python application which fills the database with data using Golemio's API[5]. However, Golemio's API was changing and the Python application at some moment was not able to support an API and put all necessary data. As a solution, I had set up an embedded H2 Database[26]. It creates a new database when Backend is started and destroys it when it is stopped. Then I created a SQL script to fill the database with temporary data, which was used for development, testing and demonstration purposes. This database is used as default, to be able to start up the server without additional configuration. Although, the configuration for the provided database is kept and can be easily turned on, instead of H2.

2.1.1 PID-Backend

2.1.1.1 Object-Relational Mapping

One of the most important parts in the software development is a work with data. The main and only source of data for PID-Backend is the Database. Therefore, it is required to properly map database data on Java objects.

Hibernate ORM[20] was used to map entity classes onto tables in the Database. I have created relevant entity classes for tables that already exist

in the database and for tables that are new. To mark classes as entities for Hibernate was used an annotations @Entity as shown on the Listing number 2.1.

```
1 @Entity
2 @Table(name = "trips")
3 @Data
4 @NoArgsConstructor
5 @AllArgsConstructor(access = AccessLevel.PUBLIC)
6 public class Trip {
7
8     @Id
9     @Column(name = "uid")
10    private String uid;
11
12    @ManyToOne
13    @JoinColumn(name = "route_id", referencedColumnName = "uid")
14    private Route route;
15    @ManyToOne
16    @JoinColumn(name = "service_id", referencedColumnName = "uid"
      , insertable = false, updatable = false)
17    @JsonIgnore
18    private Service service;
19
20    @Column(name = "shape_id")
21    private String shapeId;
22    @Column(name = "direction")
23    private int direction;
24    @Column(name = "exceptional")
25    private int exceptional;
26    @Column(name = "headsign")
27    private String headsign;
28    @Column(name = "wheelchair")
29    private boolean wheelchair;
30    @Column(name = "bikes_allowed")
31    private boolean bikesAllowed;
32    @Column(name = "block_id")
33    private String blockId;
34 }
35
```

Listing 2.1: User class example

The next will be annotating with @Column every attribute accordingly to the database table, in this case *Trips* table. Hibernate also allows to map foreign keys with @JoinColumn annotation. On the Listing number 2.1 illustrated mapping of foreign keys of tables *Routes* and *Services*.

However, I encountered problems with mapping these classes onto existing database tables, because some of the primary keys were composite. After studying documentation and its explanation on the internet I discovered a solution. In order to map such keys via Hibernate, it is required to create an additional class for the primary key and move necessary attributes from the

entity class. The new primary key should be annotated with @EmbeddedId annotation to specify to Hibernate, that it is a composite key. This is shown on the on the Listing number 2.2 and 2.3.

```
1 @Entity
2 @Table(name = "shapes")
3 @Data
4 @NoArgsConstructor
5 @AllArgsConstructor(access = AccessLevel.PUBLIC)
6 public class Shape implements Serializable {
7
8     @EmbeddedId
9     public ShapeId uid;
10
11    @Column(name = "lat")
12    private Double lat;
13    @Column(name = "lon")
14    private Double lon;
15    @Column(name = "dist_traveled")
16    private Double distTraveled;
17 }
```

Listing 2.2: Shape class example

Class ShapeId also has to be annotated with @Embedable and it has to implement Serializable interface.

```
1 @Embeddable
2 @Data
3 @NoArgsConstructor
4 @AllArgsConstructor(access = AccessLevel.PUBLIC)
5 public class ShapeId implements Serializable {
6
7     public String uid;
8     public int ptSequence;
9 }
```

Listing 2.3: User class example

Annotations *@Data, @NoArgsConstructor, @AllArgsConstructor* are Lombok[25] annotations. They describe that for this class needs to be generated getters, setters, no argument constructor and constructor with all arguments.

2.1.1.2 Delays records

By the reason of live tracking of vehicles' locations, delays, and others, the database constantly updates or substitutes this information. Therefore, *Delays records* for user favorite trips should be stored separately. Namely, there should be a task running in a different thread that checks trips in the Database and accordingly inserts new delay records. Thus, I had created such a task which starts right after the Backend is started. It periodically, every 3 minutes, fetches the favorite trips of all users. For each trip, it verifies if there is

a record of a delay for the previous stop. If there is no new delay record is saved. The pseudo code is presented on the listing 2.4.

```
1 private void updateDelays() {
2     for (Trip t : getFollowedTrips()) {
3         Vehicle v = vehicleRepo.findById(t.getUid());
4         if (v.isEmpty())
5             continue;
6         Stop lastStop = v.get().getLastStop();
7         if (delayRepo.findDelayByTripIdAndStopIdAndDate(
8                 t.getUid(),
9                 lastStop.getId(),
10                Date.Taday()).isEmpty()) {
11            addDelay(t.getUid(), lastStop.getId(),Date.Today(), v
      .getDelayLastStop());
12        }
13    }
14 }
```

Listing 2.4: Update delays task pseudo code example

In the worst-case scenario, when all trips are marked as favorite, the task will verify all of them, which might take too much time or slow down the Database. As a possible future solution, this logic can be moved into the Database. Instead of having a task in the application, database triggers can be set up. Database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. Therefore, when the table Vehicle is updated delay at the last stop can be stored in the table Delay. However, in this case, only the Database will be responsible for the delay records, which might cause other issues. For instance, if in the future the Backend will be connected to several databases, like primary and secondary, all of them have to be configured separately. This might complicate the maintenance process.

2.1.1.3 Security

One of the requirements is to support login and registration functionality to allow users to save their favorite trips and routes i.e. implement *Security* features. Since PID-Backend is a Springboot Appliaction, Springboot Security framework was chosen to manage authorization and authentication processes. In order to use it, was set up Security configuration.

First of all, I created a User entity class that implements UserDetails class from Sprinboot Security. This was necessary to make Spring use it for authorization and authentication. Additionally, was created an enum with only one role "USER". The current task does not require to support multiple roles such as Admin or Moderator. Although this is required by Spring Security and can be useful in the future.

Then I created *UserService* and *UserRepository* classes. *UserService* class must implement *UserDetailsService* from Spring Security, like in case of User class. With help of these classes, users can login and register.

PID-Backend is a stateless application i.e. it does not have, hold or memorize a state. Thus, once the request is processed, all sessions are closed. This means a user will not be able to do any operations. To solve that, can be used tokens. These tokens are generated when a user logs in and returned to the client. Then the client can use the token for operations allowed only to logged-in users. In PID-Application was used JWT Token, which I configured to make Spring use it as an authentication method.

With help of these classes, Spring can verify incoming requirements, authenticate users, and restrict access to some of the REST API.

DTO became necessary at some moment, because sometimes not all the data is needed to be transferred to the GUI and sometimes more data is required. For example when it requests information about a trip it actually needs shapes and delays as well. However, in the case of requesting user's data, all information about favorite trips is not necessary, only IDs and route names.

Implementation of other parts of PID-Backend was straightforward enough and follows the design.

2.1.2 PID-MobApp

Android development requires either to have a mobile phone with Android OS or an *Android Emulator* - Android Virtual Device (AVD). In practice it is more convenient to work with an emulator, because it is easier to debug, application can be tested for numerous different devices and no physical device is needed. However, there are some problems with running AVD on some Linux OS, including mine. For this reason, at the stage of developing PID-MobApp, I had to transfer all development on Windows OS.

In order to use *Google Maps platform* and its features, it is obligatory to create an account there. The account is needed to create an API key to obtain maps data. The downside of this approach is that for every new device on which project is built has to be assigned to the API key. This was not obvious after I transferred the project on Windows.

It was challenging to work on *XML layouts* for Activities and Fragments. Android libraries offer many different approaches to design. Most of the layouts use Constraint layout[27]. This layout allows to arrange UI objects relatively to each other. This simplifies work with a lot of items on the screen, especially in terms of different screen resolutions and sizes. Another UI element I worked with a lot is Recycler View[28]. It allows to create scrollable lists which were used in Search Results Fragment, Favourite Trips Fragment and Favourite Routes Fragment.

Here is an example of Stop Fragment layout. It is a Constraint Layout which contains two Text Views and a Recycler View. The Text Views are the names of the columns "Routes" and "Next At" on the Stop view on the figure number 1.2 (b). Under it a list of routes which is placed inside of Recycler View.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/route_name_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="@drawable/
    group_bottom_line_background"
        android:text="@string/routes"
        android:textSize="24sp"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/delay_min_tv"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/delay_min_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="@drawable/
    group_bottom_line_background"
        android:text="@string/next_at"
        android:textSize="24sp"
        android:paddingStart="8dp"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@id/route_name_tv"
        app:layout_constraintTop_toTopOf="parent" />
   <androidx.recyclerview.widget.RecyclerView
       android:id="@+id/route_rv"
       app:layout_constraintTop_toBottomOf="@id/route_name_tv"
       app:layout_constraintStart_toStartOf="parent"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

Listing 2.5: Stop Fragment Layout XML

It is being filled up with Route Item (Listing 2.6) that contains 2 Text Views. To these views Route number and arrival time are filled. In the similar fashion are used and designed other UI elements. They act like templates that are filled with required data.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="4dp"
    android:focusable="true"
    android:clickable="true"
    android:background="@drawable/selectable_under_v_23">
    <TextView
        android:id="@+id/route_name_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:padding="4dp"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/delay_min_tv"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/delay_min_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="@drawable/
    group_bottom_line_vertical_background"
        android:text="@string/next_in"
        android:textSize="16sp"
        android:paddingStart="8dp"
        android:padding="4dp"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@id/route_name_tv"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="RtlSymmetry" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Listing 2.6: Update delays task pseudo code example

The PID-MobApp has to *track vehicles' positions in real-time*. Therefore was implemented a mechanism that requests a vehicle position from PID-Backend every 5 seconds. When any vehicles appear on the map, the application starts to update them periodically via REST API. When an event that removes the vehicles' icons from the map happens, they are no longer being tracked.

2.2 Results

As a result of all work upon the system, PID-Portal was developed. It consists of three major components: Database, PID-Backend and PID-MobApp. It has

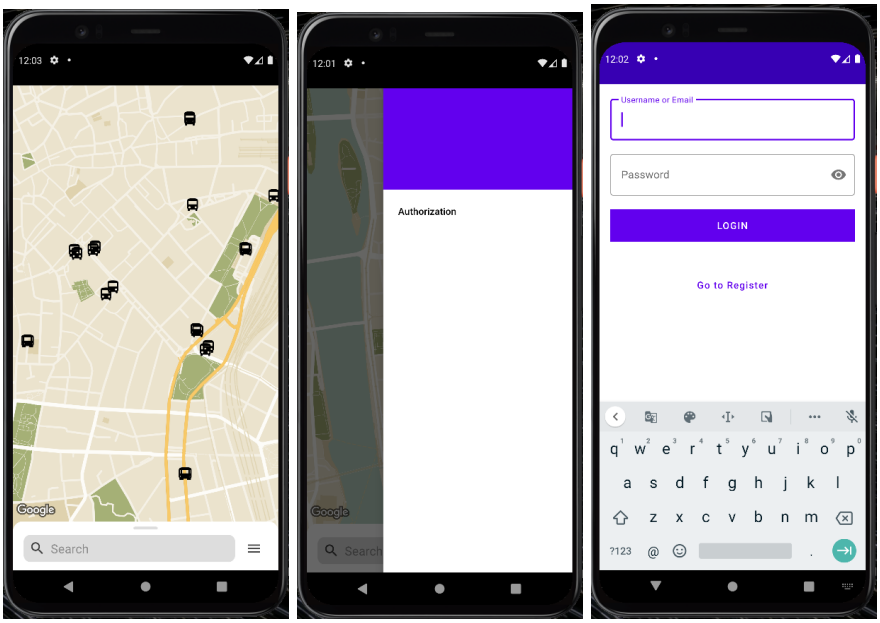

Figure 2.1: (a) Default view (b) Menu (c) Login

flexible, extendable and maintainable architecture. It follows the concept of Single responsibility. It can be comfortably scaled up horizontally or vertically.

A mobile application for Android OS was successfully developed. The Database was inherited from the supplied original database and extended for supplementary functionality. The PID-MobApp allows users to create an account and customize provided public transport data by marking or unmarking favorite routes or trips. The application provides statistics and historical data of delays for the last 7 days.

Figure 2.1 presents a login use case. It is visible that user has an option to log in or register. Once user is logged they can find favourite routes or trip as shown on the figure 2.2.

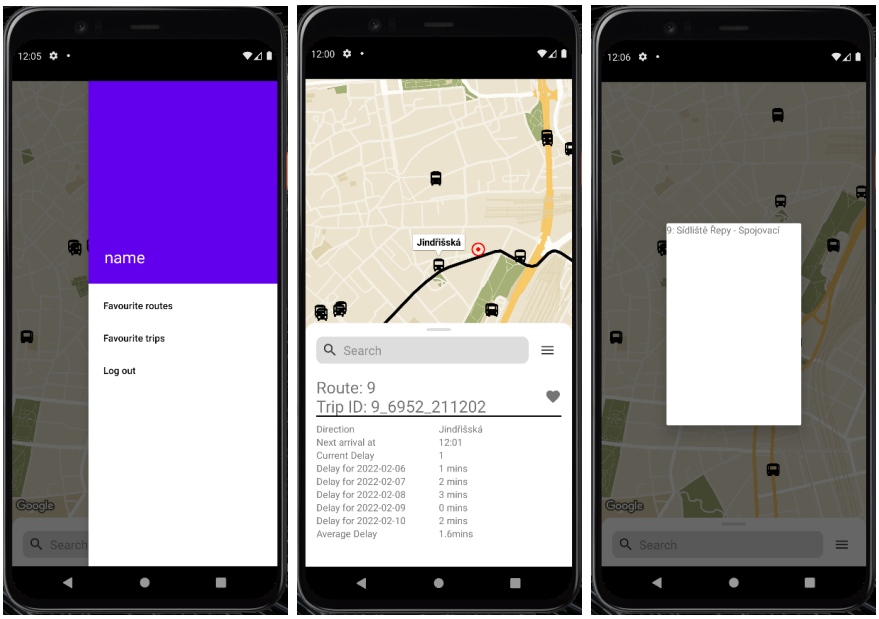

Figure 2.2: (a) Menu (b) Trip View (c) Favourite Routes

Chapter 3

Testing

An important part of any software development is testing. For this project at this stage of the development were chosen 2 types: unit testing and UI manual testing.

Unit testing is the basic first level of testing. It is performed on the functionality of individual method, preferably on interfaces. The goal of unit testing is to validate the correctness of behavior and logic of certain methods. It allows detecting errors in methods' behavior when they are changed. Which might save a significant amount of time in error finding or debugging.

In this project unit testing majorly was performed on PID-Backend because of its core-like behavior. PID-Backend contains the business logic of the whole PID-Portal, thus even slight changes have to be instantly tested and unit testing is a handy tool designed for this purpose. Although there are cases when classes are dependent and it might be not so convenient to create all classes and start the database. Therefore to solve this problem mocks were introduced. Mainly has to be tested an API of each layer of PID-Backend.

For Controllers' tests, it is necessary to check that they correctly handle input processing and call classes of the Business layer if the input is valid. Since during unit testing we do not want to create and test other classes, there were created mock classes instead.

For Services in Business Layer, it is necessary to verify that the business logic is correct and appropriate Repositories' methods are called. Services are tested in a similar manner using mocks of Repositories. Tests check responses of Services and what Repositories' methods are called.

During unit testing, problems were revealed with finding the closest trip of a given route going through a given stop. The application was finding the closest trip regardless of whether it passed the stop or not. Thus additional filtering was added. Problems with adding new favorite trips were also detected. This happened because of copying the existing method which adds favorite route and the application was trying to save favorite route instead of the trip.

In order to test the behavior of the whole PID-Portal, there were conducted manual end-to-end testing. For this testing, it was required to start up the whole system including the database. Test scenarios are derived from functional requirements to ensure that each of them is met.

As a result of manual testing were exposed a number of errors. The PID-MobApp would crush every time it would get a NULL value in a response from PID-Backend. This led to changes on both sides. Additional NULL-checks were added to PID-MobApp and NULL values are not sent from PID-Backend.

Conclusion

The main goal of this project was to create a mobile application for the visualization of public transport of the city of Prague. The result of the project was a three-layered infrastructure consisting of database, server and mobile application. The database was provided for this project as a source of data from another master's thesis. It was extended to support additional features of the application. The server was designed with REST API to provide data for the mobile application or any future GUI like web pages or IOS applications. The mobile application was created for Android OS which consumes data from the server's public API and visualizes it. The system allows users to create an account to mark favorite routes for a better user experience. Hence the goal was achieved and requirements were satisfied.

The system was designed and built to be vertically and horizontally extendable. Its architecture allows easy maintenance and develops more features. Future advancement of the system and application accordingly can be made in order to improve user experience, functionality, usability and others. Here are described the most important and big extensions, in my opinion, that can be introduced in the future.

- *Finding shortest path* is one of the features to make full use of public transport. This thesis does not focus on it in order to offer new and different functionality from alternative solutions. Although it is obviously a next thing that should be added in such application.

- *Ticket purchase* is a common option offered by public transport applications. It definitely would fit into this project as well. This change would allow users and in particular tourists, who use mostly one-time tickets instead of long-term passes, not to be dependent on shops where one-time tickets are sold. Which indirectly improves their experience of the city.

- *Different platform support* might drastically increase number of users. Such limitations like platform exclusiveness cuts down not only the number of users, but important feed back, and possible business profit. IOS application and web application are necessary additions to the system withing business context and overall availability. Multiple client support was taken into account at the design stage and already built into system. Public REST API of the server can by used by any client application.
- *Multiple city support* is another big step further to gain more users. This might cover the whole country and make user experience the same for people of different cities. Therefore it does not matter for them when they travel across the country and use public transport in different cities, because they will be using the same application.

This thesis was a good challenge and critical experience for me as a future software engineer. I have gained knowledge of working on a bigger project than I used to work on before. In the process, I have learned new technologies, approaches and best practices. My mistakes gave me a better perspective of the importance of system design, documentation, time management, constant self-improvement. I hope this project can be used as an example of this type of systems, or even a base for something bigger, where extensions described above are included.

Bibliography

[1] ICT, O. *PID Litacka [online].* [cit. 2021-10-01]. Available from: https://pidlitacka.cz/home

[2] SPOLEK, Jan. Vizualizace a predikce pražské příměstské dopravy. Praha, 2020. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[3] Seznam.cz, a.s. *PubTran [online].* [cit. 2021-10-01]. Available from: https://aplikace.seznam.cz/jizdnirady/

[4] ICT, O. *IDOS [online].* [cit. 2021-10-01]. Available from: https://idos.idnes.cz/vlakyautobusymhdvse/spojeni/

[5] O, ICT. *Golemio API [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#

[6] ICT, O. *Operator ICT [online].* [cit. 2021-10-01]. Available from: https://operatorict.cz/en/

[7] ICT, O. *Public Transport - GTFS Services [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/gtfs-services/get-gtfs-services?console=1

[8] ICT, O. *Public Transport - GTFS Routes [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/gtfs-routes/get-all-gtfs-routes?console=1

[9] ICT, O. *Public Transport - GTFS Shapes [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/gtfs-shapes/get-gtfs-shape?console=1

[10] ICT, O. *Public Transport - GTFS Vehicles [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/gtfs-stops/get-all-gtfs-stops?console=1

[11] ICT, O. *Public Transport - GTFS Vehicles [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/realtime-vehicle-positions/get-all-vehicle-positions?console=1

[12] ICT, O. *Public Transport - GTFS Trips [online].* [cit. 2021-10-01]. Available from: https://golemioapi.docs.apiary.io/#reference/public-transport/gtfs-trips/get-all-gtfs-trips?console=1

[13] Google. *SDK Platform Tools release notes [online].* [cit. 2021-10-01]. Available from: https://developer.android.com/studio/releases/platform-tools

[14] Jet Brains. *A modern programming language that makes developers happier. [online].* [cit. 2021-10-01]. Available from: https://kotlinlang.org/

[15] Google. *Welcome to Google Maps Platform [online].* [cit. 2021-10-01]. Available from: https://mapsplatform.google.com/

[16] Square, Inc. *A type-safe HTTP client for Android and Java [online].* [cit. 2021-10-01]. Available from: https://square.github.io/retrofit/

[17] Google. *Room [online].* [cit. 2021-10-01]. Available from: https://developer.android.com/jetpack/androidx/releases/room

[18] Oracle. *What is Java [online].* [cit. 2021-10-01]. Available from: https://www.java.com/en/download/help/whatis_java.html

[19] The Apache Software Foundation. *Welcome to Apache Maven [online].* [cit. 2021-10-01]. Available from: https://maven.apache.org/

[20] Red Hat. *Hibernate [online].* [cit. 2021-10-01]. Available from: https://hibernate.org/orm/documentation/5.4/

[21] VMware, Inc. *Springboot [online].* [cit. 2021-10-01]. Available from: https://spring.io/projects/spring-boot

[22] VMware, Inc. *Springboot [online].* [cit. 2021-10-01]. Available from: https://spring.io/projects/spring-security

[23] The Apache Software Foundation. *Apache Log4j 2 [online].* [cit. 2021-10-01]. Available from: https://logging.apache.org/log4j/2.x/

[24] Les Hazlewood. *JJWT [online].* [cit. 2021-10-01]. Available from: https://github.com/jwtk/jjwt

[25] The Project Lombok. *The Project Lombok [online].* [cit. 2021-10-01]. Available from: https://projectlombok.org/

[26] Mozilla Foundation. *H2 Database Engine [online]*. [cit. 2021-10-01]. Available from: `https://www.h2database.com/html/main.html`

[27] Google Developers. *Build a Responsive UI with ConstraintLayout [online]*. [cit. 2021-10-01]. Available from: `https://developer.android.com/training/constraint-layout`

[28] Google Developers. *Create dynamic lists with RecyclerView [online]*. [cit. 2021-10-01]. Available from: `https://developer.android.com/guide/topics/ui/layout/recyclerview`

Appendix A

Acronyms

API Application programming interface

GUI Graphical user interface

JSON JavaScript object notation

JWT JSON web token

ORM Object–relational mapping

OS Operating system

REST Representational state transfer

XML Extensible markup language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format