Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

# Quantification and visualization of optimum residential area clusters

Ester Atlasová

Supervisor: RNDr. Ondřej Žára
Field of study: Software
Study programme: Open Informatics
May 2022

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Atlasová** Jméno: **Ester** Osobní číslo: **483621**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Kvantifikace a vizualizace optimalních clusterů residenčních oblastí**

Název bakalářské práce anglicky:

**Quantification and visualization of optimum residential area clusters**

Pokyny pro vypracování:

Seznamte se s otevřenými daty z následujících zdrojů:
- uzly MHD Praha z GTFS dat ROPID
- soupis rezidenčních oblastí z RÚIAN a OpenStreetMap
- hustota zalidnění jednotlivých rezidenčních budov (ČSÚ)

Nad těmito daty naimplementujte model, který bude vyhledávat shluky (clustery) rezidenčních oblastí na základě následujících vstupních a omezujících podmínek:
a) počet clusterů
b) minimum pěší vzdálenosti k nejbližší zastávce MHD
c) minimální dojezdové parametry (čas/vzdálenost) k nejbližší zastávce (automobilem)

Popište, jaké existují možnosti pro hledání trasy nutné v bodě c).
Pro implementaci vycházejte ze skriptů firmy Mileus (https://github.com/mileus/analytics-scripts), konkrétně:
- compute_nstop_stats, compute_stats_for_corridor, compute_cluster_scores, select_best_cluster_for_stop
- prepare_routes_and_stops
- compute_time_matrix_for_car, compute_distance_matrix_walk

Nad tímto modelem naimplementujte interaktivní webovou aplikaci, která bude umožňovat:
1) vstup zadaných parametrů pro model,
2) vizualizaci nalezených optimálních clusterů nad mapou,
3) zobrazení hodnot (získaných i dopočítaných) pro clustery

Více než na komfort uživatelského rozhraní aplikace se soustřeďte na korektnost a obecnost rozhraní mezi klientskou částí (web) a samotným modelem (server).

Seznam doporučené literatury:

https://opendata.praha.eu/
https://github.com/mileus/analytics-scripts
https://www.geeksforgeeks.org/working-with-geospatial-data-in-python/
https://developer.mozilla.org/en-US/docs/Web

 

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**RNDr. Ondřej Žára Katedra počítačové grafiky a interakce**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

RNDr. Ondřej Žára
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.
Datum převzetí zadání

Podpis studentky

# Acknowledgements

I would like to thank RNDr. Ondřej Žára for his valuable advice and help throughout the process of writing this thesis. Also, I would like to thank my family, partner and friends for their support during my studies.

# Declaration

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

In Prague, 20. May 2022

## Abstract

This bachelor thesis aims to create a model for searching optimal residential area clusters based on user-given constraining parameters and implement an interactive web application that will take input from the user, visualize the optimal clusters on a map and visualize provided and computed values. The main focus of this work is on the communication between the web client and the model itself. Emphasis was placed on how the data will be saved and passed between the two entities and how they will be formatted.



**Keywords:** clustering, web application, geospatial data, GeoJSON, React, Flask



**Supervisor:** RNDr. Ondřej Žára



## Abstrakt

Cílem této bakalářské práce je vytvořit model pro vyhledávání optimálních shluků rezidenčních oblastí podle zadaných omezujících podmínek a naimplementovat interaktivní webovou aplikaci, která dokáže získat vstup od uživatele a vizualizovat optimální shluky na mapě a vizualizovat zadané a vypočítané hodnoty. Práce se zaměřuje na komunikaci mezi webovým klientem a samotným modelem. Důraz je kladen na to, jakým způsobem jsou data ukládaná a posílána mezi oběmi stranami a na to, jak budou formátovaná.



**Klíčová slova:** shlukování, webová aplikace, geoprostorová data, GeoJSON, React, Flask

# Contents

# Figures

# Chapter 1

# Introduction

Public transportation has always played an essential role in the way cities were formed and expanded. In 1998 an Italian physicist named Cesare Marchetti defined what is now called the Marchetti Constant. The Marchetti Constant represents the maximum time people are willing to spend commuting to work, which is thirty minutes.[36]

In the beginning, the cities were tiny, usually with a radius of around 1.5 kilometers, and expanded to a little less than a 5 kilometer radius. This is because the only way of transportation was on foot or alternatively horse-driven omnibuses, although that did not make a big difference in terms of speed. People that lived within the city borders were typically in co-living tenement apartments that could fit up to eight people in one room. On top of that, living in the center meant being exposed to various often-spread diseases due to poor hygiene conditions. As we can see, the idea of escaping into the countryside is far from new.[36]

Things changed with the invention of the steam locomotive. The first public steam railway connecting two cities was the Liverpool And Manchester Railroad, built in 1830 by George Stephenson. Not long after, cities in Europe and the United States started building such steam railways. Steam trains could carry large numbers of passengers at the speed of 16 kilometers per hour. Because of their slow acceleration, they could not stop very often, which led to building small villages around the railway stations outside of the city. These were called *railroad suburbs*.[36, 41]

Steam railways did not solve the problem of transportation inside the city center. Something more efficient than horse-lead omnibuses was needed. At the turn of the 19th and 20th century, electric streetcars started replacing the horse-powered ones. An alternative was the safety bicycle, available since the 1870s. London built the first underground line in 1863.[40, 42, 36]

Nowadays, we cannot imagine a world without cars, many households own at least one. But they were not widely affordable until Ford's Model-T introduction in 1908. The car enabled people to be independent of the established railroad suburbs. With the invention of cars came taxi services.[44]

As we can see, nowadays there is a variety of options for getting oneself from point A to point B. There are advantages and disadvantages to each mode of transportation.

The public transport services such as trains, buses, trams, etc., are cheaper, and passengers do not have to think about parking. These services tend to be very efficient in the city centers, however outside the center, they become more sparse. Therefore it can take longer to commute to the center by public transport compared to using one's car.

## 1.1 Motivation

The assignment of this thesis was inspired by a company named Mileus. Their goal is combining public transportation in the city center with taxi services in the outskirts. The main motivation is that a lot of people drive cars to the city not because of the ride in the morning, when public transport is very efficient, but for a more comfortable ride home in the evening.

Mileus is creating a service which plans a trip for the customer, in which they take a public transport vehicle from the center and somewhere along the way they are guaranteed a transfer to a taxi along the way, which will take them to the doorway of their home.[17]

## 1.2 Goals

For such intermodal service to be economically effective, it needs to start operating in geographically bound residential clusters rather than in the whole city at once. And therefore Mileus needs to analyze residential areas in order to find out where is the largest business potential and also where is the potential to increase transport serviceability by extending public transportation with on-demand services.

The optimal area for this service is a cluster of residential areas, that are not too close for walking from a public transport stop and not too far for driving from the stop. People living in the proximity of a transport station will not use this service, because they can simply walk to their doorstep and the service becomes too expensive for remote destinations.

# Chapter 2

# Analysis

In this chapter we will define the input and output for analysing residential areas and introduce the open-source data used for the algorithm, available route planning types, and the business and quantitative requirements for the final application.

## 2.1 Residential area analysis

The algorithm for analysing residential areas was implemented by Mileus and therefore its inner workings are not a part of this thesis.

The clustering algorithm takes these parameters as input:

- **Analyzed city** - defines the geography that will be analyzed
- **Analyzed public transport routes** - defines a subset of public transportation network routes that operate within the analyzed city
- **City center border polygon (optional)** - Defines the area excluded from the analysis, since network is dense in the urban center, intermodal service does not make much sense there.
- **Numerical parameters**
  - **Minimal walking distance** from a public transportation stop which is walkable and therefore residential areas located within such distance will be excluded from analysis
  - **Maximum driving distance** from a public transport stop which is yet reasonable for residents to pay the premium for taxi service on frequent basis

- **Maximum driving time** from public transportation stop, this is for the same reason as previous parameters
- **Number of analyzed consecutive public transport stops** in the route corridor, different for each route type

The result of the algorithm is an array cluster sorted by the estimated number of included residents. Each cluster includes:

- **Geography** - the convex hull of the included residential buildings
- **Included residential buildings** - coordinates of all buildings in the cluster border that satisfies the constraints
- **Excluded residential buildings** - coordinates of all buildings contained with the residential cluster that did not satisfy the constraints
- **Route id** - identification of the route that is feeding the taxi service for the residential cluster
- **Transit stops** - the stops that are feeding the taxi service for the residential cluster
- **Demographic data** - demographic data about the population of the residential cluster
- **Metrics** - reached area, reached number of residents

## 2.2 Open data

Part of the assignment was to study the open-source data used for the analysis. The data comes from four sources: ROPID, RÚIAN, Open Street Map, and ČSÚ. ROPID provides public transportation data and RUIAN, Open Street Map and ČSÚ provide data for the residentail areas and individual buildings. In this section, we will introduce these data sources and formats.

### 2.2.1 ROPID

ROPID is the primary organizer for public transportation in Prague and its surroundings. They provide open data, including regularly-updated timetables, real-time coordinates of transportation vehicles and statistical data. The offered datasets are mainly for data analysts and software application developers. [18, 21]

The data is in the GTFS (or General Transit Feed Specification) format, created by the company Google and used for its Transit trip planner. GTFS lets public transport organizers publish data in a widely-used format for software applications. The basic GTFS format, also known as GTFS static, includes information regarding the timetables and geography. Many applications also use the real-time extension, consisting of arrival predictions, current vehicle positions, and service advisories. GTFS data is a collection of text files collected into a zip file. These files have information about a specific aspect of public transportation. [32, 33, 35]

Files that are in the GTFS dataset for Prague are the following: [37]

- agency.txt - includes all the public transport companies operating in Prague and its surroundings
- stops.txt - consists of all the stops that are currently in use and their geological location
- stop_times.txt - contains the time of arrival and departure of each vehicle at individual stops in its route
- routes.txt - currently operating routes
- trips.txt - all the trips planned on each currently operating route
- calendar.txt - an array of ones and zeros for each service_id, the service_id defines a group of trips and is referenced in trips.txt[1]

### 2.2.2 RÚIAN

RÚIAN is the territorial identification, address, and property registry of the Czech Republic. It is a public list that does not include personal information and is unique source of addresses not only for the public administration.[15]

RÚIAN provides this list in multiple ways and formats:

- VDP - an online application enabling users to look into and extract data from the RÚIAN registry
- GML[2] format - a file format for expressing geographical features
- CSV format[3] - a data file, where each record is a separate row, and each record may have one or multiple fields which are separated by a comma

[1]The array defines whether the service group is operating on given day of the week or not, zero means the group is not operating, one means the group is operating

[2]Geography Markup Language `https://www.ogc.org/standards/gml`

[3]Comma-Separated Values `https://datatracker.ietf.org/doc/html/rfc4180`

Each record consists of:

- municipality
- municipal district - if the municipality is divided into these districts
- street
- house number
- postal code
- x-coordinate in the S-JTSK[4] system
- y-coordinate in the S-JTSK system
- date from when this record is valid

### 2.2.3 Open Street Map

Open street map is an open-source map. Anyone can edit the map and access the underlying map data. Data can be exported directly from the map. However, this has a limit to the size of the exported area.[1]

For more extensive datasets, there are plenty of options to choose from:[14]

- *BBBike* - exports user-chosen bounding boxes
- *Geofabrik* - exports specific administrative units
- *Planet OSM* - can export the entire OSM dataset
- *Overpass API* - convenient for querying the OSM dataset

OSM data is exported into an XML[5] file, an image, or HTML code. We will focus on the XML file format.[8]

XML is a file format based on tags used to store, search and share data. The tags can be nested and create a tree-like structure. XML does not define specific tags. The Open Street Map XML file has three tags at the top of the tree structure called elements. These tags are *node*, defining a point in space, *way*, defining linear features and area boundaries, and *relation* used to explain how other elements work together.[6]

[4] Coordinate system for the Czech Republic and Slovakia `https://epsg.io/2065`
[5] Extensible Markup Language

Buildings in OSM are defined as polygons using the *way* tag, this tag has the polygon nodes as its children and information about the building, like the type of building or number of levels and flats. This information is in a tag consisting of a key and value. To filter the buildings that are residential, the *building* key must have one of the following values: *apartments, detached, terrace, semi_detached house, hut, ger, houseboat, static_caravan or house.*

### 2.2.4 ČSÚ

ČSU is the primary statistical office in the Czech Republic. It collects, analyzes, and publishes statistical information for the state and local authorities and the public and foreign institutions. The ČSÚ organization compiles information about the demographical and economic growth in the Czech Republic, processes the results of local government council and national government elections and European Parliament elections, and organizes the Czech Republic Census every ten years.[46]

The important dataset for the clustering algorithm is the list of cadastral municipalities in Prague and their corresponding population density. The most recent one published is from the year 2014.[45] For each cadastral area there are two items, the population number and the area size in hectares.

## 2.3 Route planning types

To decide, whether a building is close enough to satisfy the two numerical parameters, maximal driving distance and maximal driving time, a route planning service is necessary. This section describes available open-source route planning services.

- **Open Source Routing Machine** is a router created to use data from the Open Street Map project. It uses a technique called contraction hierarchies to find the shortest path, which is used mostly in car-navigation systems, web-based route planners, traffic simulation, and logistics optimization.[19, 3]

- **Open Route Service** provides multiple different service, all based on the geographic data from Open Street Map. This service can compute ono-to-one, one-to-many and many-to-many routes for supported modes of transport. The Mileus Residential area analysis uses this service.[20]

- **GraphHopper**is an efficient routing library and sever written in Java. It can use different algorithms such as Dijkstra, A*, or Contraction Hierarchies.[13]
- **Valhalla** is a routing engine used in the Mapzen and Mapbox services and SDKs.[24]

## 2.4 Requirements

In this section, you will find business, functional and non-functional requirements. These requirements were defined based on the needs of Mileus. Some were also added by the author.

### 2.4.1 Business requirements

Business requirements define what behavior is expected from the application and why it is expected.

- BR01 - As a user, I need to be able to select the analyzed city to receive its city model (coordinates, public transport network) and select which city will be analyzed
- BR02 - As a user, I need to be able to choose from the available routes in the city's public transportation network, to select the ones that will be analyzed
- BR03 - As a user, I need to be able to define the city center border, to define which public transport stops will be excluded from the analysis
- BR04 - As a user, I need to be able to enter the numerical parameters to set constraints for the residential buildings
- BR05 - As a user, I need to be able to assign the name of the created analysis to identify it between multiple analysis' tasks quickly
- BR06 - As a user, I need to be able to pick which analysis results I want to visualize
- BR07 - As a user, I need to be able to choose which particular cluster data I want to which particular cluster's data and metrics I want to visualize

### 2.4.2 Qualitative requirements

Qualitative requirements are based on the qualities and characteristics that are desired from the system.

- QR01 - client application will be interactive
- QR02 - client application will be functional in multiple modern browsers
- QR03 - client application will be optimized for speed
- QR04 - the system will be scalable for other cities

# Chapter 3

# Design

The design of the system was created based on the business and qualitative requirements. This chapter will introduce all the components of the system, set the functional requirements, present the data design, the REST API and client application and look at how data is sent between components.

## 3.1 System components

The whole system includes five components:

- **Client application** - collects parameters from user and visualizes the results
- **Application server (REST API)** - includes classes and functions for serving data to the client application, receiving requests from the client application with data for the clustering analysis, and communicates with the Mileus server to run the analysis
- **Mileus server (REST API)** - includes classes and functions for running the analysis and saving the results into the Mongo Database
- **Mongo Database** - includes a list of available cities for analysis, city models (coordinates, public transport routes), information about all the previously ran analysis tasks (name, start time, end time, status, numerical parameters) and the analysis results
- **PostgreSQL Database** - includes the static city data used for the analysis, used only by the Mileus server

The following diagram shows how data is send between the components. The PostgreSQL database is not shown, as it is only used by the Mileus server and therefore not part of this thesis.

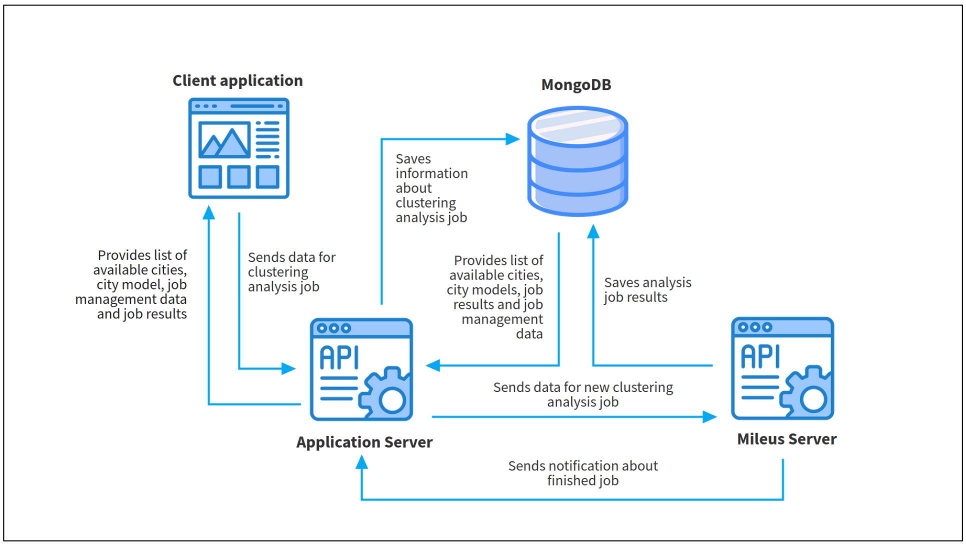


**Figure 3.1:** Diagram showing components of the system, designed using icons from Flaticon.com

## 3.2 Functional requirements

Functional requirements define what is expected from the implementation and constrain the scope of the system. They are based on business requirements and qualitative requirements.

- **FR01** - The application server needs to send analysis input data to the Mileus server
- **FR02** - The application server needs to provide analysis' results data to the client application
- **FR03** - application server needs to be connected to a database
- **FR04** - The application server needs to serve data for creating a new analysis task to the client application
- **FR05** - The application server needs to serve data about previously created analysis tasks to the client application
- **FR07** - The application server needs to receive analysis input data from the client application

- **FR08** - The client application needs to provide a user interface for choosing the analysis input parameters
- **FR09** - The client application needs to provide a user interface showing a list of information about previously created analysis tasks
- **FR10** - The client application needs to provide a user interface for visualizing the analysis results

## 3.3 REST API

The client application communicates with the application server through a REST API.

An API[1] is a *set of definitions and protocols for building and integrating application software.*[27] An API serves as a mediator for requesting and providing information. It defines what data is required from the information consumer and what data is required from the information provider. It is a great tool for organisations or other service providers to share information, while maintaining control over security and authentication. [27]

REST[2] is a set of constraints that define and architectural style. Data and functionality are called resources and are connected to one United Resource Identifier (also URI). Resource are accessed and modified in a consistent approach, most widely used are HTTP methods (POST, GET, PUT, DELETE) which roughly correspond to CRUD operations (CREATE, READ, UPDATE, DELETE).[34]

Our application uses REST API for creating and reading resources. The resources of our application and their corresponding URIs are:

- **City list** - available cities for analysis
  - `/api/city-list`
- **City model** - the center coordinates and routes of a city
  - `/api/city-model`
- **Job information** - list with id, name, start and end time and status of each analysis job
  - `/api/job-information`

[1] Application Programming Interface
[2] Representational State Transfer

- **Job result** - resulting clusters of clustering analysis
  - `/api/job`

## 3.4 User interface design

The user interface was designed into three parts, the job overview page, pages for creating new job, and pages with job results visualization.

### 3.4.1 Job overview page

The job overview page serves as the application's main page and gives the user an option to look at the results of jobs that were already created or create a new job. The already created jobs are shown in a table, and to go to the job creation pages, the page includes a *new job* button.

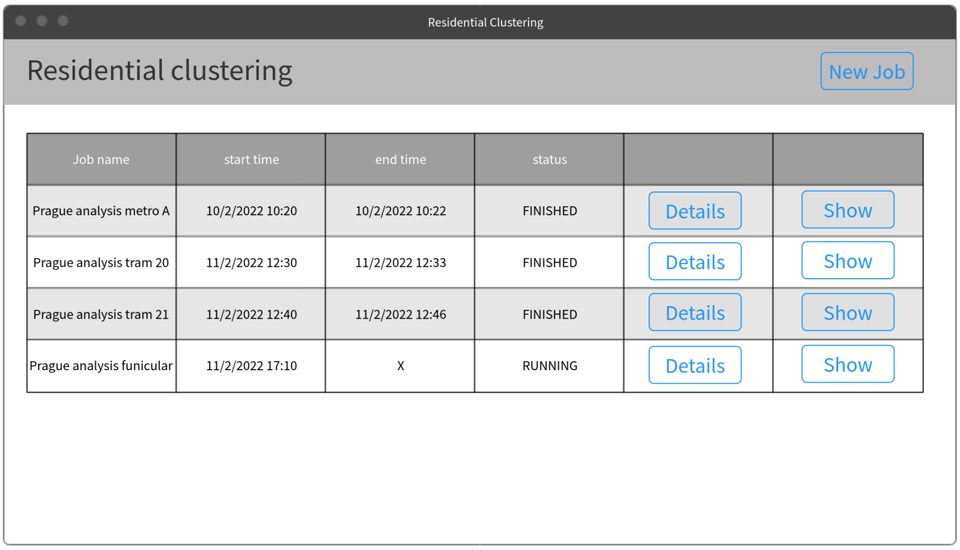


**Figure 3.2:** Job overview page

### 3.4.2 Pages for creating new job

The clustering analysis input data (see section 2.1) is divided into four parts. Based of these parts four pages were designed. The pages are in the order in which they appear when the user is choosing parameters for the new job. Each page has a *back* button taking the user to the previous page. The first three pages have a *next* button taking it to the following page and the fourth page has a *start job* to start the job.

WireframePro was used to create the page designs[3]

## City picker

As we can see in figure 3.3 this page includes only a drop down menu with the available cities. The user must chose a city to go to the following page.

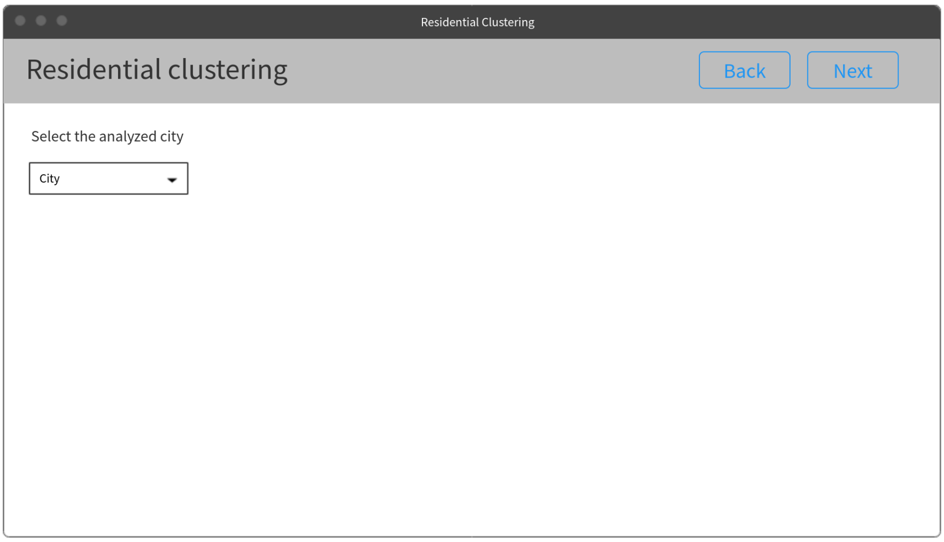


**Figure 3.3:** City picker page

## Public transport routes picker

As we can see in figure 3.4 this page lets the user pick one or more routes from accordions on the left. Each chosen route is visualized on the map. The user must choose at least one route to go to the following page.

## City center picker

As we can see in figure 3.5 this page contains a map with the routes chosen in the previous page. This page lets the user draw a polygon representing the city center. Since this parameter is optional, the user can go to the next page without drawing the polygon.

[3] https://www.mockflow.com/apps/wireframepro/

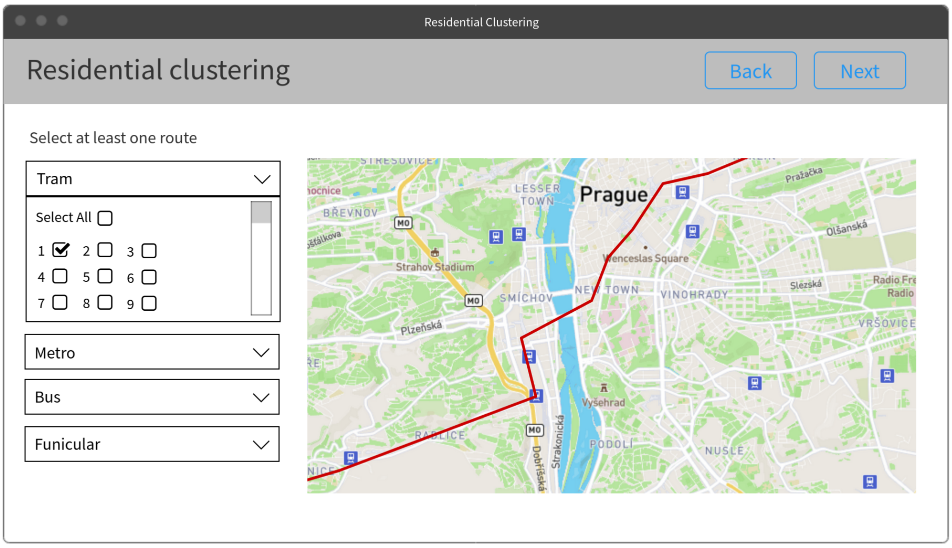


**Figure 3.4:** Public transport routes picker page design, designed using map tiles from Mapbox and OpenStreetMap

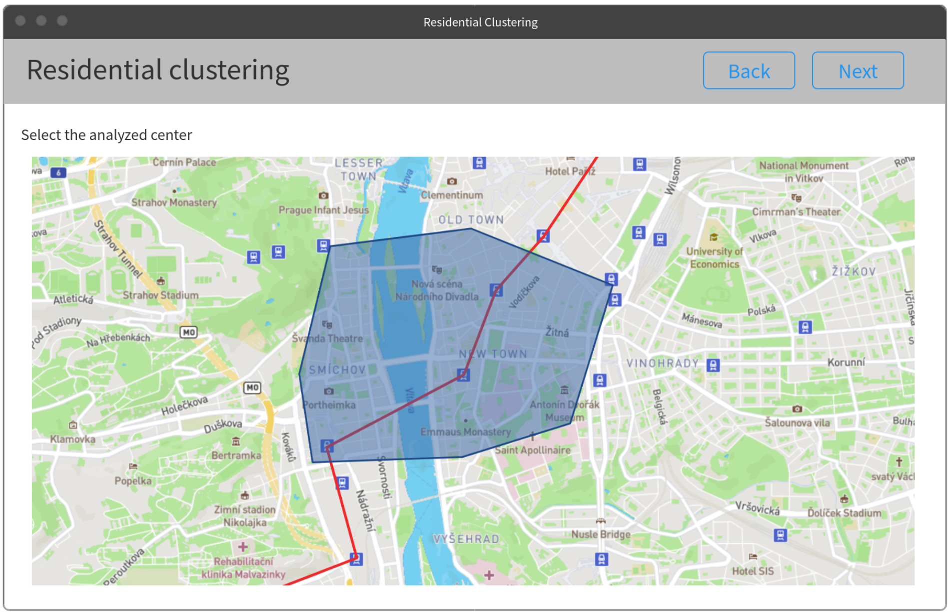


**Figure 3.5:** City center picker page design, designed using map tiles from Mapbox and OpenStreetMap

## Numerical parameters form

The last page in this section contains a form for the numerical parameters for the analysis and job name and number of histograms, as seen in figure 3.6. All the inputs except job name must be filled in with numerical values and the job name input is not constrained. All the inputs must be filled in to let the user start the job. When the user clicks the start job button, a modal is shown asking them if they want to proceed to start the job. After proceeding the user is taken to the job overview page where the newly created job is already in the table with the status `RUNNING`.

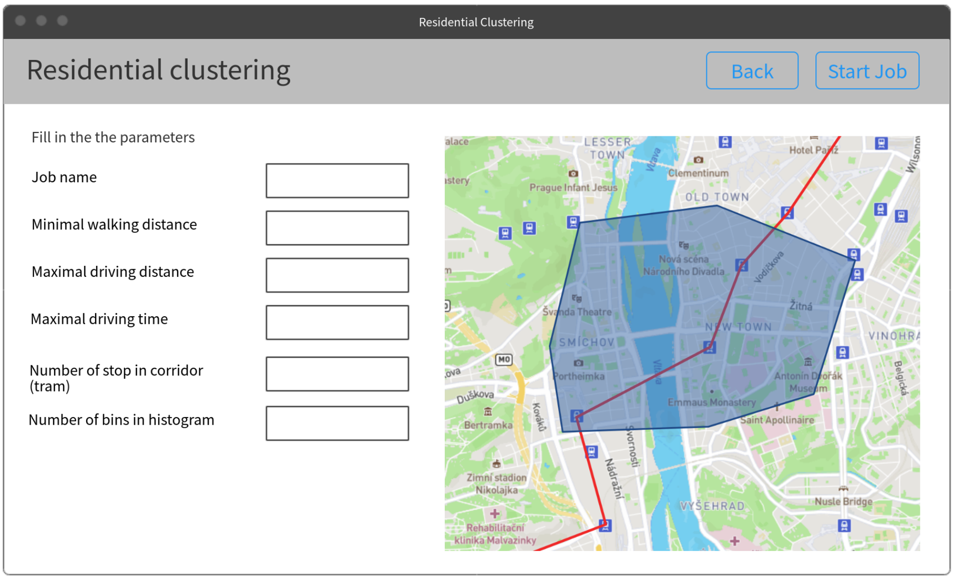


**Figure 3.6:** Numerical parameters form page design, designed using map tiles from Mapbox and OpenStreetMap

## 3.4.3 Visualization pages

We designed two pages for the results' visualization. The first is for visualizing all the clusters, and the second page shows more details about a particular cluster. Further on, the first page will be referred to as the *high-level visualization page* and the second as the *detailed visualization page*.

## 3.4.4 High-level visualization page

This page shows a map with all the clusters and two buttons, *back* and *details*. The back button takes the user to the main page. The details button opens a tab component showing the numerical parameters for the analysis and the calculated reached residents and area of each cluster. The map also lets the user choose the number of best clusters shown with an input control. The page design is shown in figure 3.7

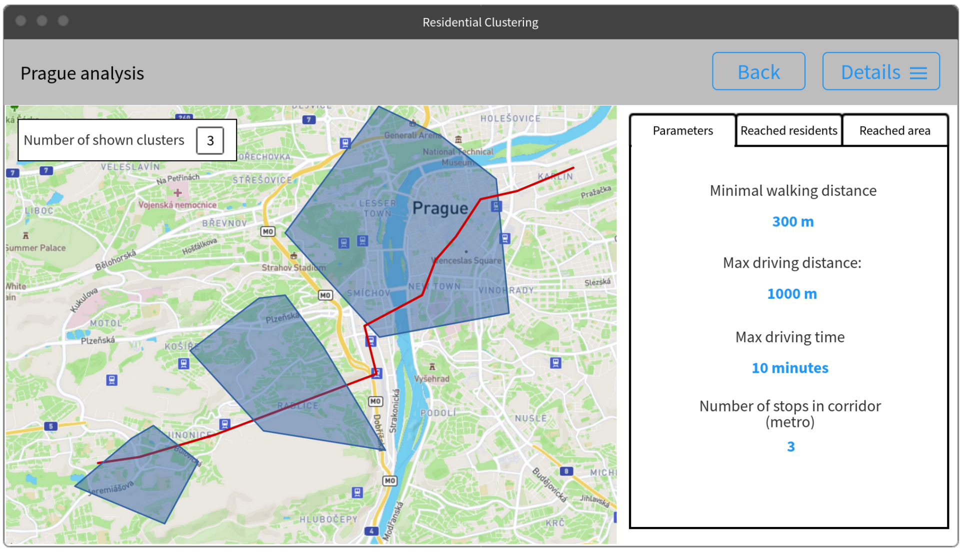


**Figure 3.7:** High-level visualization page design, designed using map tiles from Mapbox and OpenStreetMap

## 3.4.5 Detailed visualization page

The detailed visualization page is similar to the high-level visualization page. It also shows a map and a *back* and *details* button. The map shows the visualization of one cluster, the back button takes the user to the high-level visualization page and the details button opens a tab component with graphs giving the user more information about this cluster. The page design is shown in figure 3.8

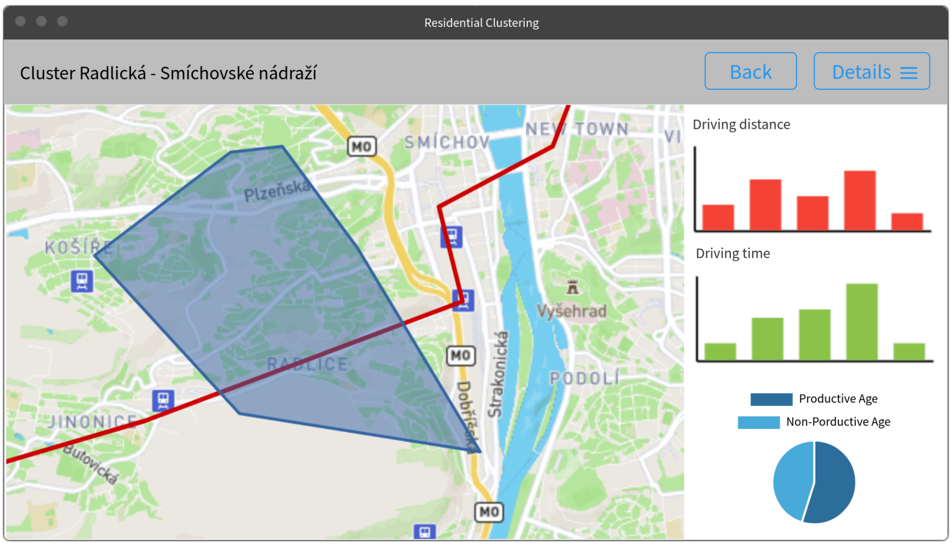


**Figure 3.8:** Detailed visualization page design, designed using map tiles from Mapbox and OpenStreetMap

## 3.5 Communication between the application server and the client application

In this section describes how the client application communicates with the application server through the REST API.

- `GET /api/job-information` called by the main page to receive information about all the jobs that have been created
- `GET /api/cities` called by the city picker page to receive a list of available cities
- `GET /api/city-model/{selectedCity}` called by the public transportation page to receive static data of a particular city identified by the `selectedCity` parameter
- `POST /api/job` called by the numerical parameters form page to create a new analysis job
- `GET /api/job/{jobId}` called by the high level visualization page to receive the results of a particular analysis job identified by the `jobId` parameter

## 3.6 Communication between the application server and the Mileus server

There were more proposals for designing the process of creating a new analysis job and saving the results. Since the calculations can last longer than a couple of minutes, creating a new job and saving the results were separated into two requests. First, application server sends a request for creating a new job to the Mileus server. The Mileus server then replies with the job id. The results data size could potentially be very big. To avoid sending large files through a request, the Mileus server saves them to the Mongo database. After that, the Mileus server sends a request to application server with the job id. This request serves as a notification for the application server that the job has finished. The application server replies to this request with the job id. As a result, both servers need to be connected to the same Mongo database, as we can see in figure 3.1. The interface for the Mileus server is also a REST API. The URIs and their corresponding HTTP methods are the following:

- `POST /v0/clusters/new-job` (in the Mileus server REST API) creating a new job

- `POST /api/job/{job-id}` (in the application REST API) notifying the application server about a completion of a job, the job is identified by the `job-id` parameter

## 3.7 Data design

The data used and created in this application is in the JSON format and saved into a database. JSON is a standard text-based format for representing structured data based on JavaScript object syntax.[26] There are 4 resources, available cities, city models, job information and jobs. This section introduces the structure of this data.

**Available cities**

- an array of strings, each string is a city name

**City models** - each city is one object which comprises of:

- name of the city
- city model
  - center coordinates - latitude and longitude of the city
  - available public transport routes - grouped into route types, each route is connected to its linestring

**Job Information** - each job is one object which comprises of:

- job id
- start timestamp - time and date when the job started
- end timestamp - time and date when the job ended
- status - either `RUNNING` or `FINISHED`

**Jobs** - each job object contains:

- job id
- job name

- city name
- center coordinates - latitude and longitude
- numerical parameters - as seen here 2.1
- number of bins in histograms
- clusters - array of cluster object, the structure of the cluster object is shown below

**Cluster object**:

- cluster id
- boundary - a polygon in the GeoJSON format which will be introduced in this section 4.1.7
- included residential buildings - a set of points in the GeoJSON format
- excluded residential buildings - a set of points in the GeoJSON format
- route linestring - a linestring in the GeoJSON format
- corresponding public transport stops - name, latitude and longitude
- demographic data - productive age distribution
- metrics - reached residents, reached area

# Chapter 4

# Implementation

This chapter will introduce the technology and libraries that were used to develop the application and show how the implementation fulfilled the business, functional and qualitative requirements.

The client application was written in the JavaScript programming language, and the server in Python. JavaScript is a *lightweight, interpreted, object-oriented language*[31]. Most people know JavaScript as the scripting language for web pages. Therefore JavaScript enables a web page to have dynamic content, like interactive maps or videos. Apart from object-oriented programming, JavaScript supports procedural and functional programming.[31]

Python is a *interpreted, interactive, object-oriented programming language*[10] that also supports procedural and functional programming styles. Python can solve various problems, such as machine learning, developing web applications, and creating graphical user interfaces.[10]

## 4.1 Used technology and libraries

### 4.1.1 React.js

ReactJS is a popular open-source *JavaScript library used for buildings user interfaces*[22]. Thanks to the popularity of React, many third-party libraries are available to use. Instead of separating technologies, HTML and JavaScript, React separates concerns by dividing the interface into loosely coupled components. React uses JSX to create elements and renders them into the DOM. JSX is a syntax extension that allows writing HTML in JavaScript.[22, 16]

DOM represents web documents with a logical tree. Each branch of the tree ends with a node, and each node contains an object. DOM provides methods allowing programs to change the document's structure, style, and content.[5]

Angular is another technology often used for web development, a TypeScript_based programming framework. It includes an extensive collection of well-integrated libraries and is an excellent tool for creating scalable web applications.[25]

As the author did not have much experience with either of these technologies, the documentation and learning time was the main factor in deciding which one to choose. Angular solves many problems at once.[38] Therefore it has a higher learning curve than React. Also, React has excellent documentation with tutorials. That is why React was chosen for the development of the client application.

### 4.1.2 Leaflet

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. It has all the basic mapping features necessary for creating interactive maps and can be extended with many plugins. Leaflet was chosen as the mapping library for the client application because of its well-written documentation and a large number of plugins. [28] The leaflet-draw plugin was used for selecting the center border.

For using Leaflet in React, there is a library react-leaflet. This library provides React components for Leaflet layers.[30] Both Leafet and React-Leaflet were used in the application. At the beginning of the development, maps were created using the Leaflet library, taking advantage of its great documentation. React-Leaflet provided better tools for creating a control form on the page for the detailed visualization page.

Leaflet includes:[43]

- Map element
- UI elements - Marker, Popup, Tooltip
- Raster layers - TileLayer, ImageOverlay
- Vector layers - Polygon, Polyline, Rectangle, CircleMarker
- Grouped types - GeoJSON, FeatureGroup, LayerGroup
- Controls - Zoom, Layers

### 4.1.3 Recharts

Recharts is one of React's oldest and most reliable chart libraries. It was built with React and D3[1]. This library supports Scalable Vector Graphics (or SVG) and uses declarative components.[23]

Recharts were used for visualizing the statistical data on the detailed visualization page. The library needed to provide event listeners to handle mouse events for the charts to be interactive. Recharts fulfilled this requirement. Several other interactive chart libraries are available for React, for example, Victory.js, visx, or react-vis.[29]

### 4.1.4 Material UI

Material UI is the React version of Google's Material Design component library. Material UI used it to create a unified design throughout the client application. The library offers a complete set of tools for creating customizable and reusable components, like buttons or accordions, for faster development.

### 4.1.5 Webpack

Webpack is used to prepare the application for running in the browser. It takes all the application modules and bundles them into static assets, which can be run in the browser. Webpack supports all browsers which are ES5-compliant.[2, 7]

### 4.1.6 CORS

The application server and the client application both run on different ports and thanks to CORS, they are able to share resources to one another. CORS is an HTTP-header based mechanism that lets a server define for which foreign origins a browser is permitted to load resources.[4]

### 4.1.7 Geojson

The cluster and center boundaries, route lines, residential buildings, and public transport stops visualized in the maps are encoded as geometry types

[1] Data-Driven Document

in the GeoJSON format. GeoJSON is a JSON-based format for geospatial data. Each geometry object is a feature and includes the geometry data and other optional properties. A group of features is a FeatureCollection. The supported geometry types are Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon. In our application, boundaries are represented as Polygons, route lines as LineStrings, and buildings and stops as Points.[11, 12]

An example of a GeoJSON object is the following.

```
{
    "type": "FeatureCollection",
    "Features":[{
          "type": "Feature",
          "geometry": {
            "type": "Point",
            "coordinates": [
                50.071259750369975, 14.403881527676178
            ]
          },
          "properties": {
            "name": "Anděl"
        }
    }]
}
```

### 4.1.8 Flask

The server developed at Mileus, runs code implemented in Flask. For future purposes, it was appropriate to stay consistent and implement the application server code also in Flask. Flask is a web framework for developing web applications in python. It is a microframework. This means its core is designed to stay simple and scalable. Flask does not include an abstraction layer for database support but instead supports corresponding extensions. Another popular web development framework in python is Django.[9]

### 4.1.9 Python libraries

When providing analysis results' data, the application server needs to prepare them for the client application. The application server used Python libraries numpy, pandas, shapely and gtfsk. Shapely was used to add cluster center and bounds to adjust the map in the detailed visualization page. Numpy and pandas were used to calculate the cluster histograms. Lastly, the application server used gtfsk, numpy, and pandas to prepare Prague's static city model data and add the route linestrings to the results.

### 4.1.10 MongoDB

The system data is in JSON format. Therefore the system required a document database. MongoDB is a NoSQL document database that can store high volumes of data. Mongo does not store data in tables, and rows like traditional relational databases do. Instead, it stores data in collections and documents. Documents consist of key-value pairs. Collections include sets of documents and functions. The Python library pymongo was used To connect to our database from the flask server.[39]

## 4.2 User Interface implementation

In section 3.4 we designed the appearance of the user interface pages. The final pages were implemented based on these designs and technologies discussed in section 4.1. The application has seven pages in total.

### 4.2.1 Job overview page

The `JobOverview.js` module implements the job overview page. It is divided into three components, header, table, and dialog. The table includes job information and details and shows buttons. The details button opens a dialog component with the numerical parameters of the analysis job. When the job status is `RUNNING`, then the show button is deactivated. The implementation of this page can be seen in figure 4.1 and the dialog for showing the numerical parameters in figure 4.2.

### 4.2.2 City Picker page

The City Picker page is implemented by the `CityPicker.js` module. It includes a header and dropdown select. The next button is activated only when a city has been chosen. If the select is closed without selecting a city, a red alert shows up asking the user to select a city. The implementation of this page is shown in figures 4.3, 4.4 and 4.5.

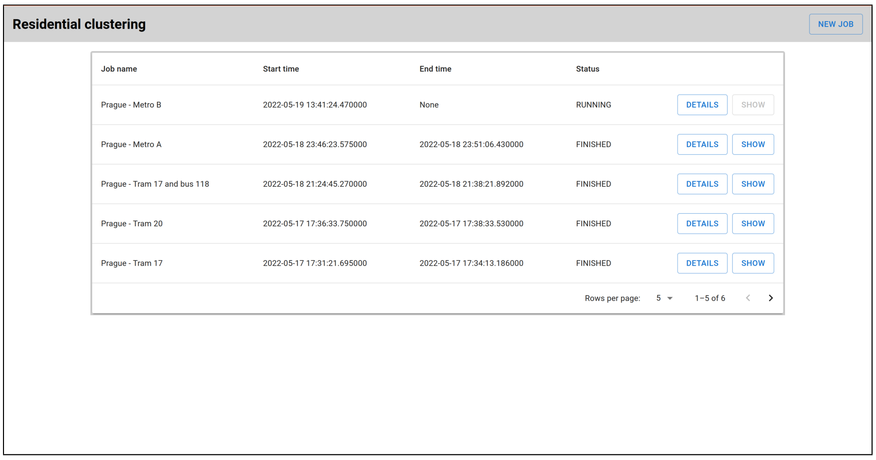


**Figure 4.1:** Job overview page implementation

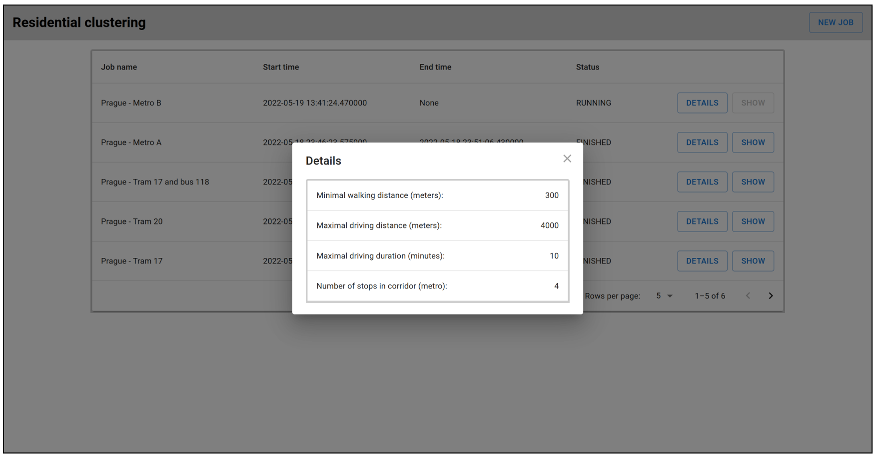


**Figure 4.2:** Job details dialog implementation


**Figure 4.3:** City picker page implementation


**Figure 4.4:** City picker page implementation with alert message


**Figure 4.5:** City picker page implementation with chosen city

### 4.2.3 Public Transport Routes Picker Page

The public transport route picker page is provided by `RoutePicker.js`. This page contains a header, an accordion group and a map. A route can by chosen by opening one of the accordions and selecting the route checkbox. After selecting the route its linestring appears on the map. The user must select at least one route to activate the next button. The implementation of this page can be seen in figures 4.6 and 4.7.

### 4.2.4 City Center Picker page

The city center picker page is implemented in the `CenterPicker.js` module. It includes a map with the previously chosen routes and controls for drawing a polygon, which will define the city center border. The controls include three

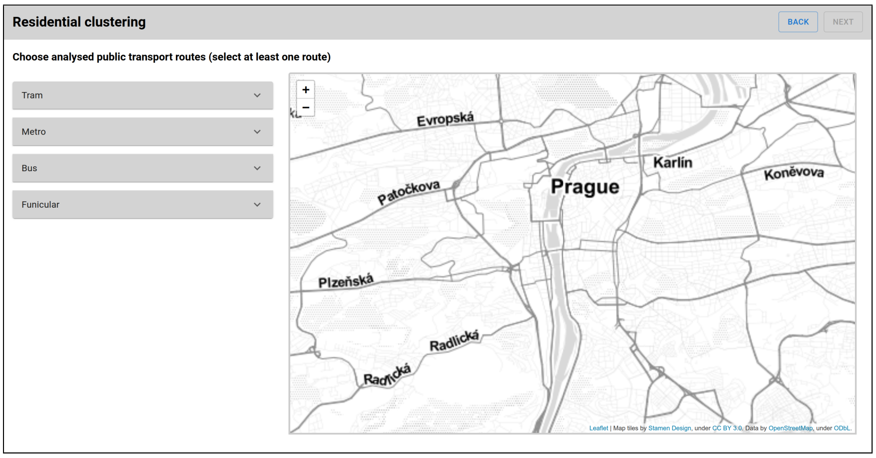


**Figure 4.6:** Public transport route picker page implementation, without a selected route

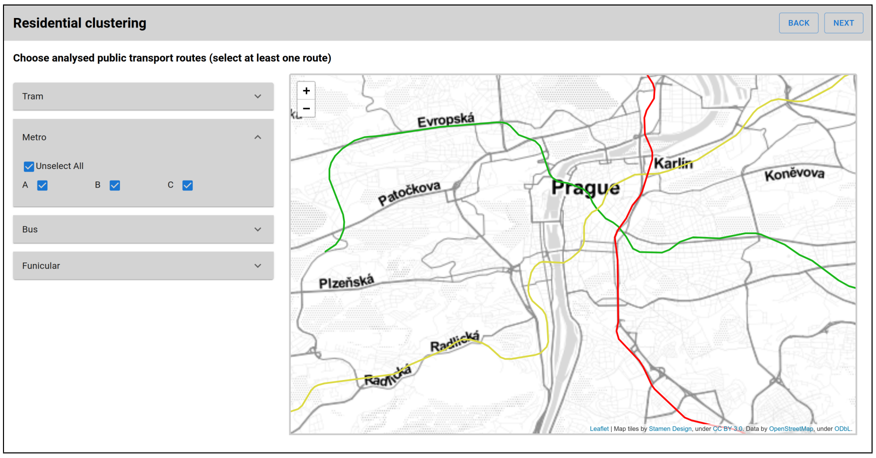


**Figure 4.7:** Public transport route picker page implementation, with all metro routes selected

buttons, one for drawing, one for editing and one for deleting the polygon. The next button is always active, as this is an optional parameter. The page implementation can be seen in this figure 4.8.

### 4.2.5 Numerical Parameters Form Page

The page for filling in the numerical parameters is implemented in the `ParametersFormPage.js` module. As designed in 3.4.2, this page contains text field inputs for the numerical parameters, the job name and the number of bins in the histogram. All the inputs must be filled in according to the constraints defined in the design. If a text field is left unfilled or in the wrong format, a red alert message shows up. The implementation of this page is shown in figures 4.9, 4.10, 4.11 and 4.12.

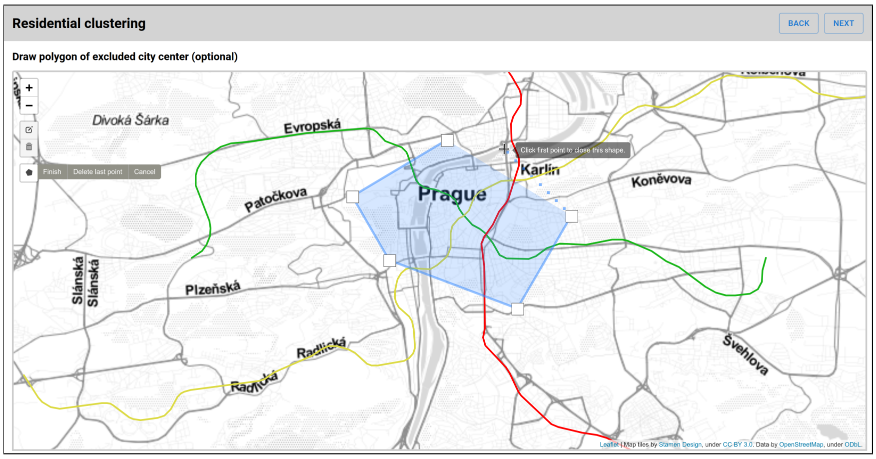


**Figure 4.8:** City center picker page implementation with polygon being drawn by the user

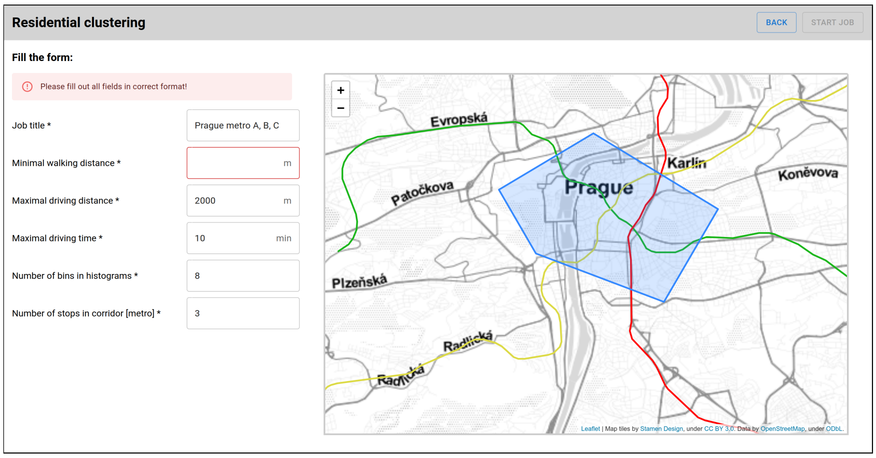


**Figure 4.9:** Numerical parameters form page implementation, with an input left unfilled

## 4.2.6 High-level Visualization page

Based on the design 3.4.4, this page includes a header with buttons back and details, a map with the visualized cluster data, routes and transport stops, and a tab component, which is opened when the details button is clicked. When the back button is clicked, the user is taken to the job overview page. The Map also includes a text field input control, which lets the user define how many clusters will be visualized on the map. The clusters are sorted by the number of reached residents.

When the cursor hovers over a cluster a tooltip with the cluster name shows up and the cluster is highlighted. When the cursor hovers over the cluster in the reached residents tab, only the corresponding cluster is shown on the map. The high-level visualization page is implemented in the `HighLevelViz.js`

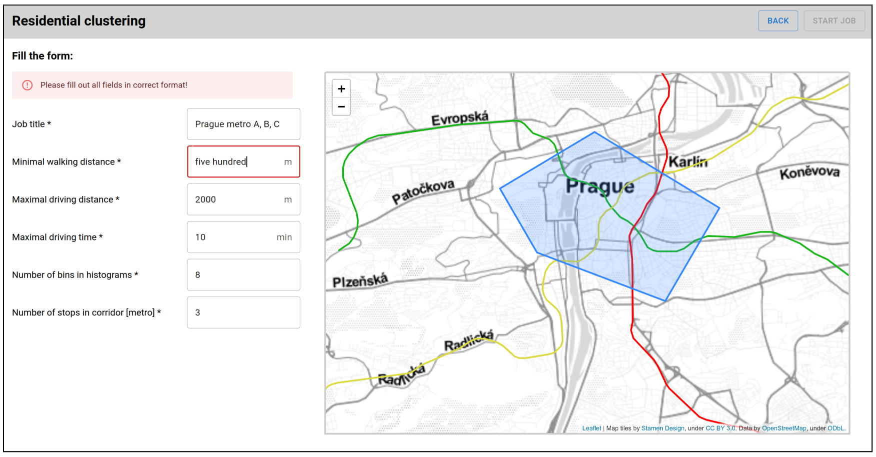


**Figure 4.10:** Numerical parameters form page implementation, with an input filled in the wrong format

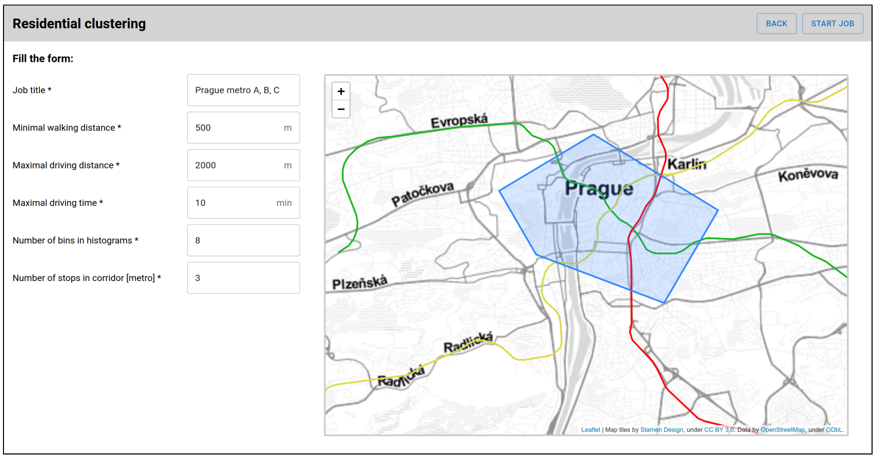


**Figure 4.11:** Numerical parameters form page implementation, with all input filled correctly

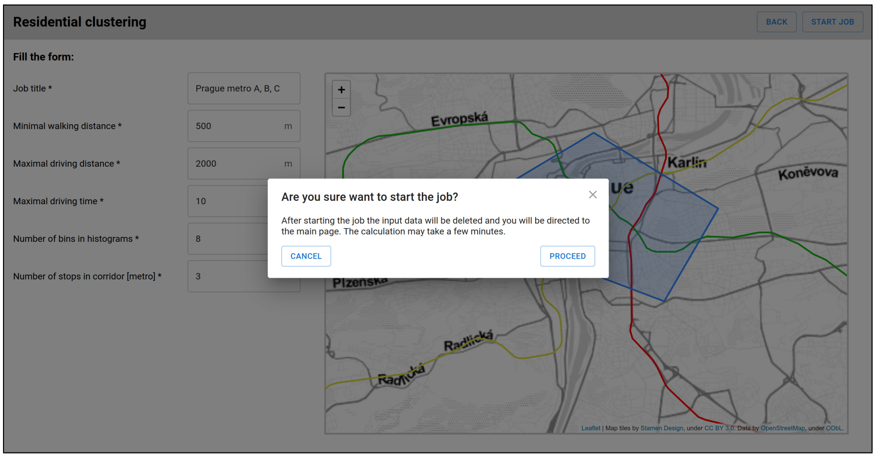


**Figure 4.12:** Start job dialog implementation

module and is shown in figures 4.13, 4.14 and 4.15.

There are two ways the user can enter the detailed visualization page. Either by clicking on the cluster in the map, or by clicking on the cluster in the reached residents tab.

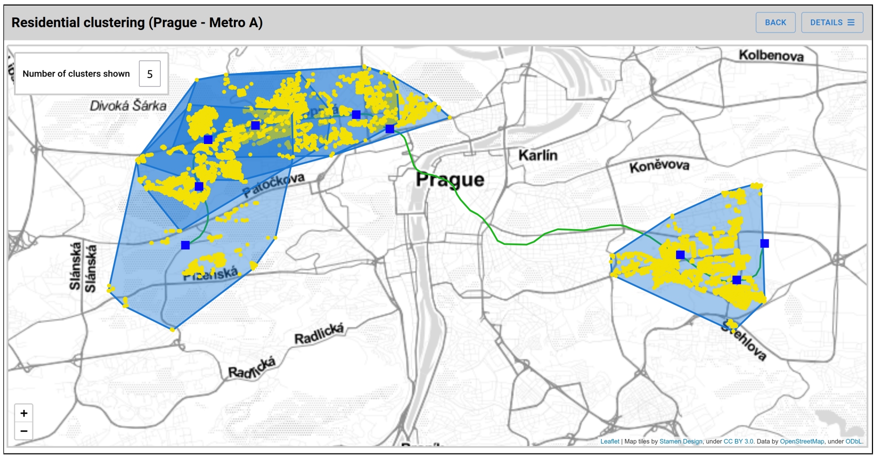


**Figure 4.13:** High-level visualization page

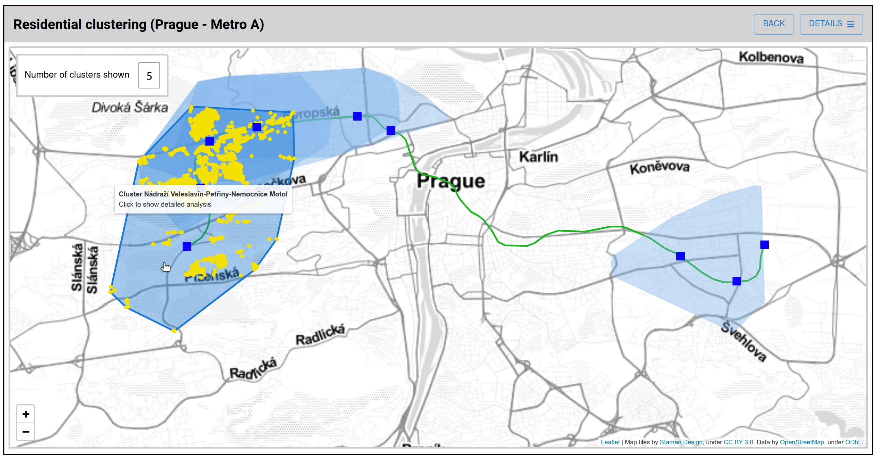


**Figure 4.14:** High-level visualization page, mouse hovered over cluster in map

## 4.2.7 Detailed Visualization page

Finally, the detailed visualization page is implemented by the `DetailedViz.js` module. This page includes two maps with the visualized cluster, a header with back and details button. The back button takes the user to the high level visualization page and the details button opens a tab with histograms, showing the distribution of residential buildings based on taxi ride distance and duration, and a pie chart with the productive age ratio.

The first map visualizes the residential buildings in different colors depend-

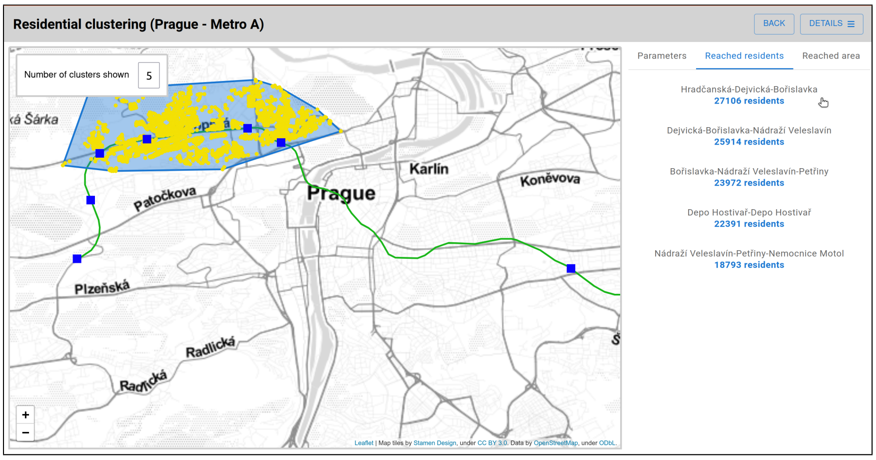


**Figure 4.15:** High-level visualization page, mouse hovered over cluster in reached residents tab

ing on how far they are from the closest public transport stop. The second map visualizes the included buildings by yellow points and excluded buildings by grey point. To switch between these maps there is a radio button control in the left upper corner of the map.

The implementation of this page is shown in figures 4.16, 4.17, 4.18 and 4.19.

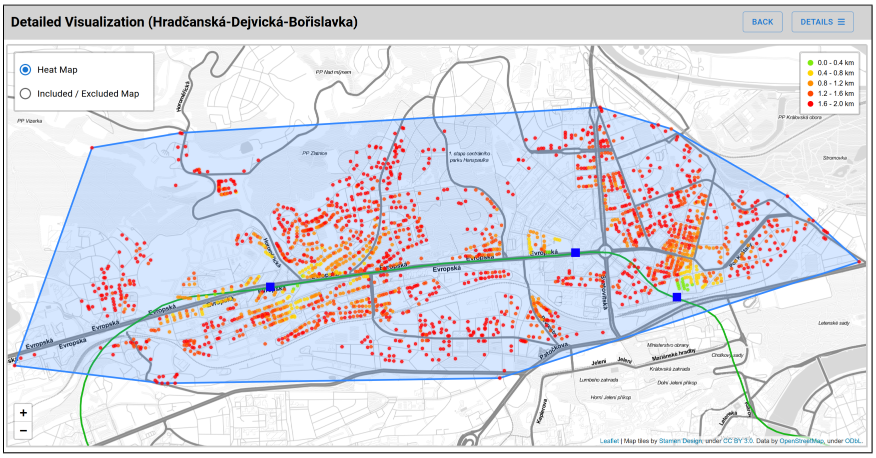


**Figure 4.16:** Detailed visualization page

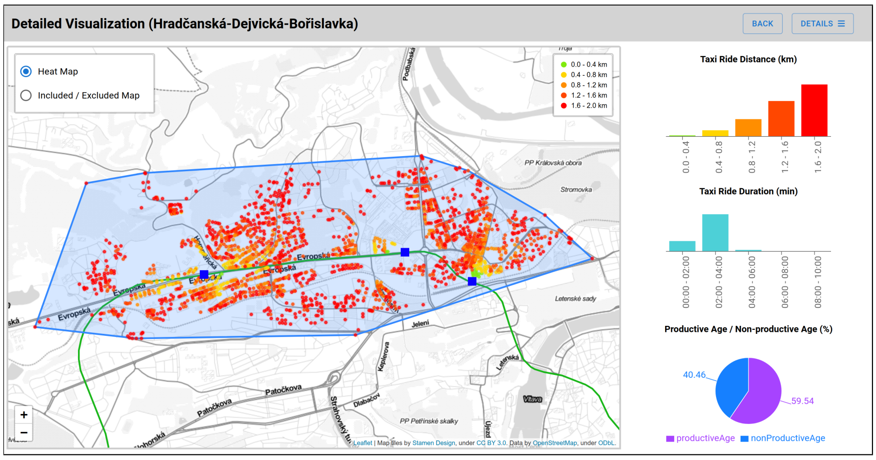


**Figure 4.17:** Detailed visualization page, residential buildings colored according to the distance from the closes public transport stop

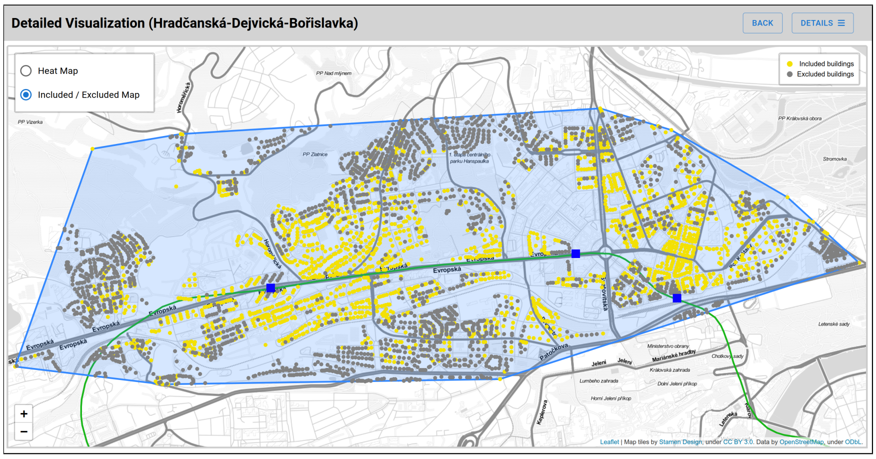


**Figure 4.18:** Detailed visualization page, residential buildings colored yellow if included, grey in excluded

## 4.3 Server implementation

The server side of this application was implemented based on the designed REST API and data structure from chapter 3. All the classes and methods for handling requests from the client application are implemented in `app.py`. The functions for database operations are implemented in `db_utils.py`. Lastly, the functions that handle preparing data for the client application and for Mileus server are implemented in `data_prepare.py`. In this section, the implementation of the http methods for each resource will be explained.

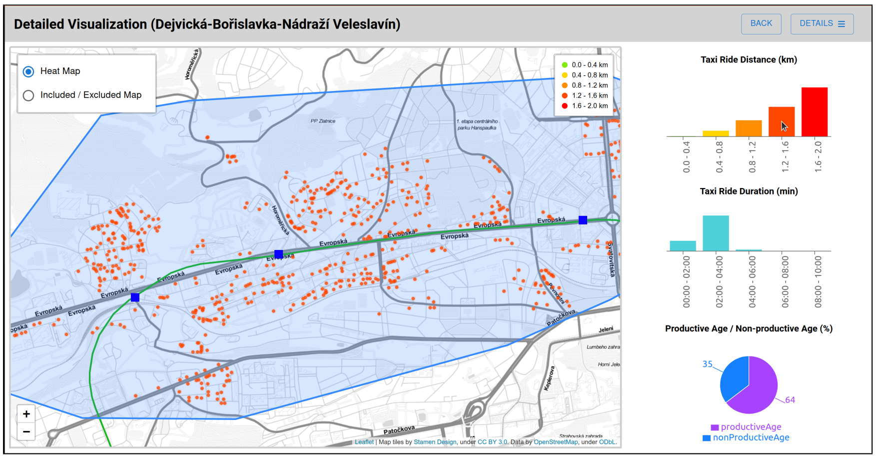


**Figure 4.19:** Detailed visualization page, cursor hovered over the 1.2 - 1.6km area showing only the residential buildings from this group on the map

### 4.3.1 `GET /api/cities`

This request is handled by the `AvailableCitiesController` class. After the request is received, the list of available cities is found in the database and sent as the response.

### 4.3.2 `GET /api/city-model/{selectedCity}`

This request is handled by the `CityModelController` class. When the request is received the city model of the selected city is found in the database and sent as the response.

### 4.3.3 `GET /api/job-information`

The GET request for the job information resource is handled by the `JobInformationController` class. Because the parameters data is saved in the results collection, both the job information list and results list have to be retrieved from the database. The response is a list of object, each object including job information and parameters of the corresponding job.

### 4.3.4 `POST /api/job`

The request to start a new job is handled by the `JobListController` class. After receiving this request the controller:

1. Sends the necessary input data that came with the request to the Mileus server, to start the calculation, which responds with the job id
2. Saves the job start timestamp
3. inserts the job id, job name, start timestamp and status, into the job information database collection
4. inserts the job id, job name, job parameters, city name and city coordinates into the job results database collection

### 4.3.5 `GET and POST/api/job/{jobId}`

These requests are handled by the `JobController` class.

After the GET request is received the the job data is found by job id, processed and send as the response. The data is processed to add the name, bounds, and center, histograms and route linestrings to each cluster.

When receiving the POST request, the job end timestamp is saved and the corresponding job information end time and status is updated.

# Chapter 5

# Testing

The scope of this application was defined by the business and functional requirements. To decide, whether the application fulfilled these requirements, user inputs and expected application behaviour were defined and compared with the implemented output. According to the testing scenarios, all the business and functional requirements were met. The testing scenarios can be found in Appendix B.

The qualitative requirements define the characteristics of the system. The first qualitative requirement was having an interactive client application. The interactive elements of our system are:

- a linestring is drawn on map after user selects a route
- user can draw the city center polygon
- user can define how many cluster they want to see on the map
- when user hovers over a cluster it is highlighted and a tooltip with the cluster name is shown
- when user hovers over a cluster in the reached residents tab, only the cluster is shown in the heat map
- when user hovers over a residence distance group in the Taxi Ride Distance histogram, the other groups are hidden

The second qualitative requirements was *client application will be functional in multiple modern browsers.* The application was tested on these browsers: Mozilla Firefox, Chromium, Brave and Google Chrome. It is functional on all of them.

The third qualitative requirement was *client application will be optimized for speed*. Unfortunately, the application is too slow because the data loading speed is not optimized. Future development should focus on this drawback.

The last qualitative requirement was *the system will be scalable for other cities*. To add a new city, data for the city model object for this city would have to be processed and added to the database and the city name string added to the cities resource.

# Chapter 6

# Conclusion

The goal of this bachelor thesis was to design and implement an application for collecting user input and visualizing the result of a residential area clustering analysis. The development went through all the software development process steps. The application is aimed at providing an interactive tool for Mileus' clustering analysis. Therefore they were in the role of the client and defined the requirements for this application. The system was designed and implemented based on these requirements. All the defined functional and business requirements were fulfilled and most of the qualitative as well.

An important part of this process was defining the structure of the input and output data for the calculation, since the two servers were implemented by different people. This enabled the two servers to develop independently.

The contribution of this bachelor thesis is creating an interactive tool, that can help decide which residential areas have the biggest potential for services that aim at connecting public transport networks with on-demand service providers.

## 6.1 Future development

As discussed earlier in 5, the biggest drawback of this application is slow data loading. This should be solved in the future. This application could also allow the user to upload and download a file with analysis results.

The application has not been deployed, since the Mileus server is in a private repository. For now it can be run locally. For the purposes of using this application for making business decisions it would be better to deploy it, as more people could open the results visualization and analyze them.

# Appendix A

# Bibliography

[1] About openstreetmap. https://wiki.openstreetmap.org/wiki/About_OpenStreetMap. Accessed: 2022-12-05.

[2] Concepts. https://webpack.js.org/concepts/. Accessed: 2022-18-05.

[3] Contraction hierarchies.

[4] Cross-origin resource sharing (cors). https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. Accessed: 2022-18-05.

[5] Document object model (dom)ng jsx. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model. Accessed: 2022-18-05.

[6] Elements. https://wiki.openstreetmap.org/wiki/Elements. Accessed: 2022-12-05.

[7] Esmascript 5 compatiblity table. https://kangax.github.io/compat-table/es5/. Accessed: 2022-18-05.

[8] Export. https://wiki.openstreetmap.org/wiki/Export. Accessed: 2022-12-05.

[9] Foreword. https://flask.palletsprojects.com/en/2.1.x/foreword/. Accessed: 2022-18-05.

[10] General information. https://docs.python.org/3/faq/general.html#what-is-python. Accessed: 2022-17-05.

[11] Geojson. https://geojson.org/. Accessed: 2022-18-05.

[12] The geojson format. https://datatracker.ietf.org/doc/html/rfc7946. Accessed: 2022-18-05.

[13] Graphhopper. https://wiki.openstreetmap.org/wiki/GraphHopper. Accessed: 2022-18-05.

[14] How to download dataset from openstreetmap? https://towardsdatascience.com/beginner-guide-to-download-the-openstreetmap-gis-data-24bbbba22a38. Accessed: 2022-12-05.

[15] Informace o rÚian. https://www.cuzk.cz/ruian/RUIAN/Informace-o-RUIAN.aspx. Accessed: 2022-12-05.

[16] Introducing jsx. https://reactjs.org/docs/introducing-jsx.html. Accessed: 2022-18-05.

[17] Mileus, commuting home. comfortably. https://mileus.com/. Accessed: 2022-05-05.

[18] O organizaci ropid. https://pid.cz/o-organizaci/o-organizaci-ropid/. Accessed: 2022-11-05.

[19] Open source routing machine. https://wiki.openstreetmap.org/wiki/Open_Source_Routing_Machine. Accessed: 2022-18-05.

[20] Openrouteservice. https://github.com/GIScience/openrouteservice. Accessed: 2022-18-05.

[21] Otevřená data pid. https://pid.cz/o-systemu/opendata/. Accessed: 2022-11-05.

[22] React. https://reactjs.org/. Accessed: 2022-18-05.

[23] Recharts. https://github.com/recharts/recharts. Accessed: 2022-18-05.

[24] Valhalla. https://wiki.openstreetmap.org/wiki/Valhalla. Accessed: 2022-18-05.

[25] What is angular? https://angular.io/guide/what-is-angular. Accessed: 2022-18-05.

[26] Working with json. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. Accessed: 2022-18-05.

[27] What is a rest api? https://www.redhat.com/en/topics/api/what-is-a-rest-api, 2020. Accessed: 2022-14-05.

[28] Vladimir Agafonkin. Leaflet, an open-source javascript library for mobile-friendly interactive maps. https://leafletjs.com/index.html. Accessed: 2022-18-05.

[29] Ilana Brudo. Top 11 react chart libraries. https://www.tabnine.com/blog/top-11-react-chart-libraries/, 2020. Accessed: 2022-18-05.

[30] Paul Le Cam and contributors. Introduction. https://react-leaflet.js.org/docs/start-introduction/. Accessed: 2022-18-05.

[31] MDN contributors. About javascript. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. Accessed: 2022-17-05.

[32] Google. Gtfs realtime overview. https://developers.google.com/transit/gtfs-realtime. Accessed: 2022-12-05.

[33] Google. Gtfs static overview. https://developers.google.com/transit/gtfs. Accessed: 2022-12-05.

[34] Lokesh Gupta. Rest architectural constraints. https://restfulapi.net/rest-architectural-constraints/, 2022. Accessed: 2022-14-05.

[35] Chris Harrelson. Happy trails with google transit. https://googleblog.blogspot.com/2006/09/happy-trails-with-google-transit.html, 2006. Accessed: 2022-12-05.

[36] Jonathan English. The commuting principle that shaped urban history. https://www.bloomberg.com/news/features/2019-08-29/the-commuting-principle-that-shaped-urban-history, 2019. Accessed: 2022-05-05.

[37] Dopravní podnik hl. m. Prahy a.s. Praha gtfs. https://transitfeeds.com/p/praha/801. Accessed: 2022-12-05.

[38] João Reis. Angular vs react: a comparison of both frameworks. https://www.imaginarycloud.com/blog/angular-vs-react/, 2020. Accessed: 2022-18-05.

[39] David Taylor. What is mongodb? introduction, architecture, features example. https://www.guru99.com/what-is-mongodb.html. Accessed: 2022-18-05.

[40] The Editors of Encyclopaedia Britannica. bicycle, vehicle. https://www.britannica.com/technology/bicycle. Accessed: 2022-05-05.

[41] The Editors of Encyclopaedia Britannica. George stephenson, british inventor. https://www.britannica.com/biography/George-Stephenson. Accessed: 2022-05-05.

[42] The Editors of Encyclopaedia Britannica. streetcar. https://www.britannica.com/technology/streetcar. Accessed: 2022-05-05.

[43] OpenStreetMaps contributors Vladimir Agafonkin. Leaflet api reference. https://leafletjs.com/reference.html. Accessed: 2022-18-05.

[44] Wikipedia contributors. Ford model t. https://en.wikipedia.org/w/index.php?title=Ford_Model_T&oldid=1088267980. Accessed: 2022-05-05.

[45] Wikipedie. Obyvatelstvo podle 112 katastrálních území hl. m. prahy. https://www.czso.cz/documents/11236/17812557/CR_L4_KU.xlsx/231bacd8-d7c4-4ed2-83e5-d41d7e4a0e3f?version=1.5, 2014. Accessed: 2022-13-05.

[46] Wikipedie. Český statistický úřad — wikipedie: Otevřená encyklopedie. https://cs.wikipedia.org/w/index.php?title=%C4%8Cesk%C3%BD_statistick%C3%BD_%C3%BA%C5%99ad&oldid=21240649, 2022. Accessed: 2022-13-05.

# Appendix B

# Testing scenarios

| | |
|---|---|
| Test ID | 01 |
| Prerequisites | User is on the job overview page |
| User input | User opens clicks on the "new job" button |
| Expected output and behaviour | The city picker page is opened with the next button deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 02 |
| Prerequisites | User is on the city picker page |
| User input | User opens the dropdown select and closes it without selecting a city |
| Expected output and behaviour | A red alert shows up above the dropdown select with the message *Analysed city wasn't selected!* and the next button remains deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 03 |
| Prerequisites | User is on the city picker page |
| User input | User opens the dropdown select and selects "Prague" |
| Expected output and behaviour | The "Next" button is activated |
| Result | success |

| | |
|---|---|
| Test ID | 04 |
| Prerequisites | User is on the city picker page |
| User input | User opens the dropdown select, selects "Prague" and clicks on "Next" |
| Expected output and behaviour | The public transport route picker page is opened with the "Next" button deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 05 |
| Prerequisites | User chose the city "prague" and is on the public transport route picker page |
| User input | User opens the "metro" accordion and selects the checkbox labeled "A" |
| Expected output and behaviour | A green linestring is drawn on the map and the "next" button is activated |
| Result | success |

| | |
|---|---|
| Test ID | 06 |
| Prerequisites | User chose the city "prague" and is on the public transport route picker page |
| User input | User opens the "metro" accordion and selects the checkbox labeled "Select All" |
| Expected output and behaviour | All three checkboxes "A", "B" and "C" are checked and three linestrings, one green, one red and one yellow, are drawn on the map and the next button is activated |
| Result | success |

| | |
|---|---|
| Test ID | 06 |
| Prerequisites | User chose the city "prague" and is on the public transport route picker page |
| User input | User opens the "tram" accordion, selects the checkbox labeled "17" and clicks on "next" |
| Expected output and behaviour | The city center picker page is opened with a blue linestring drawn on the map and the next button is active |
| Result | success |

| | |
|---|---|
| Test ID | 07 |
| Prerequisites | User chose the city "prague" and is on the public transport route picker page |
| User input | User opens the "metro" accordion, selects the checkbox labeled "Select All" and then selects the check box "Deselect All" |
| Expected output and behaviour | No checkboxes are checked, no linestrings are drawn on the map and the next button is deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 08 |
| Prerequisites | User chose the city "prague", metro "A" and is on the city center picker page |
| User input | User clicks on the "next" button |
| Expected output and behaviour | The numerical parameters form page is opened with a green linestring and without a polygon on the map, and the "start job" button is deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 09 |
| Prerequisites | User chose the city "prague", metro "A" and is on the city center picker page |
| User input | User clicks on the draw control on the left side of the map, draws a polygon and clicks on "next" |
| Expected output and behaviour | The numerical parameters form page is opened with a green linestring and without a polygon on the map, and the "start job" button is deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 10 |
| Prerequisites | User chose the city "prague", metro "A" and is on the numerical parameters form page |
| User input | User fills in these values into the text fields: ["test", "200", "2000", "5", "7", "3"] |
| Expected output and behaviour | The "start job" button is activated |
| Result | success |

| | |
|---|---|
| Test ID | 11 |
| Prerequisites | User chose the city "prague", metro "A" and is on the numerical parameters form page |
| User input | User fills in these values into the text fields: ["test", "a", "2000", "5", "7", "3"] |
| Expected output and behaviour | The "start job" button is deactivated and a red alert is shown above the form with the message: *Please fill out all fields in correct format!* |
| Result | success |

| | |
|---|---|
| Test ID | 12 |
| Prerequisites | User chose the city "prague", metro "A" and is on the numerical parameters form page |
| User input | User clicks on the "Maximal driving distance" texfield input and then clicks outside |
| Expected output and behaviour | The "start job" button is deactivated and a red alert is shown above the form with the message: *Please fill out all fields in correct format!* |
| Result | success |

| | |
|---|---|
| Test ID | 13 |
| Prerequisites | User chose the city "prague", metro "A" and is on the numerical parameters form page |
| User input | User fills in these values into the text fields: ["test", "200", "2000", "5", "7", "3"] and clicks on "start job" |
| Expected output and behaviour | A dialog is opened with the message: *Are you sure want to start the job? After starting the job the input data will be deleted and you will be directed to the main page. The calculation may take a few minutes.* and two buttons: "Cancel" and "Proceed" |
| Result | success |

| | |
|---|---|
| Test ID | 14 |
| Prerequisites | User chose the city "prague", metro "A", filled in these values into the numerical parameters form ["test", "200", "2000", "5", "7", "3"], clicked "Start job" and the start job dialog is open |
| User input | User clicks on "Proceed" |
| Expected output and behaviour | The overview page is opened and the job with these values ["test", the current time, with a small delay, "NONE", "RUNNING"] is displayed as the first row in the job overview table and has the "show" button deactivated |
| Result | success |

| | |
|---|---|
| Test ID | 15 |
| Prerequisites | User is in the job overview page |
| User input | User clicks on "Show" button in the row with the job name "Prague Metro A" |
| Expected output and behaviour | The high level visualization page is opened with 5 clusters on the map and a control for selecting the shown number of clusters |
| Result | success |

| | |
|---|---|
| Test ID | 16 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job |
| User input | User changes the "5" to "3" in the control for changing the amount of clusters shown |
| Expected output and behaviour | Only 3 clusters are shown on the map |
| Result | success |

| | |
|---|---|
| Test ID | 17 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job and filled in the number of shown clusters control with "1" |
| User input | User hovers over the cluster |
| Expected output and behaviour | A tool tip with "*Cluster Hradčanská-Dejvická-Bořislavka*" is shown |
| Result | success |

| | |
|---|---|
| Test ID | 18 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job |
| User input | User clicks on "Details" |
| Expected output and behaviour | A tab component is shown on the left with the values ["200m", "10 minutes", "2000m"] |
| Result | success |

| | |
|---|---|
| Test ID | 19 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job and clicked on "Details" |
| User input | User clicks on the "Reached area" tab |
| Expected output and behaviour | The "Reached area" tab opens and shows the values ["8.69 km2", "9.76 km2", "8.82 km2", "6.55 km2", "13.31 km2"] |
| Result | success |

| | |
|---|---|
| Test ID | 20 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job and clicked on "Details" |
| User input | User clicks on the "Reached residents" tab |
| Expected output and behaviour | The "Reached residents" tab opens and shows the values ["27106 residents", "25914 residents", "23972 residents", "22391 residents", "18793 residents"] |
| Result | success |

| | |
|---|---|
| Test ID | 21 |
| Prerequisites | User is in high level visualization page of the "Prague Metro A" job and filled in the number of shown clusters control with "1" |
| User input | User clicks on the cluster |
| Expected output and behaviour | The detailed visualization page is opened with the cluster buildings draw in different colors and in the left upport corner there is a radio button control with "Heat Map" selected |
| Result | success |

| | |
|---|---|
| Test ID | 22 |
| Prerequisites | User is in detailed visualization page of cluster "Dejvická-Bořislavka-Nádraží Veleslavín" from the job "Prague Metro A" |
| User input | User clicks on "Included / Excluded Map" |
| Expected output and behaviour | The radio button "Included / Excluded Map" is selected, the map changes to buildings colored yellow and grey |
| Result | success |

| | |
|---|---|
| Test ID | 23 |
| Prerequisites | User is in detailed visualization page of cluster "Dejvická-Bořislavka-Nádraží Veleslavín" from the job "Prague Metro A" |
| User input | User clicks on "Details" |
| Expected output and behaviour | A tab opens on the right with 2 histograms and 1 pie chart, the pie chart values are [40, 59] |
| Result | success |

| | |
|---|---|
| Test ID | 24 |
| Prerequisites | User is in the job overview page |
| User input | User clicks on "Details" in the row job name "Prague - Tram 17" |
| Expected output and behaviour | A dialog opens up with a table and these values [300, 1000, 5, 3] |
| Result | success |