

Hochschule für Technik

Stuttgart University of Applied Sciences

Photogrammetry and Geoinformatics



Hochschule
für Technik
Stuttgart

GIS Studio

Online Map Creation for Population Growth Analyzing

(24 June 2024)

Submitted by:

Md Ashifur Rahaman (20015301)

Fatemeh Rafiei (20015299)

Md. Raisul Islam (20015091)

Table of Contents

1. Introduction.....	4
1.1 Background Information.....	4
1.2 Objectives	4
1.3 Significance of the Study.....	4
1.4 Project Relevance	5
1.5 Innovation and Technological Integration.....	5
2. Literature Review.....	6
2.1 Open Data Utilization.....	6
2.1.1 Open Data for Research	6
2.1.2 Case Studies in Open Data	6
2.2 Population Growth Analysis.....	6
2.2.1 Demographic Analysis Techniques.....	6
2.2.2 GIS and Population Studies.....	6
2.3 Application of Population Projection Techniques	6
2.3.1 Exponential Population Projection.....	7
2.3.2 Innovations in Population Projection	7
2.4 Technologies for Data Management and Visualization.....	7
2.4.1 Flask	7
2.4.2 PostgreSQL	8
2.4.3 pgAdmin.....	8
2.5 Online Mapping and Visualization Tools.....	9
2.5.1 MapLibre.....	9
3. Methodology	10
3.1 Study Design	10
3.2 Materials	10
3.3 Procedure.....	10
3.3.1 Data Collection and Preparation:	10
3.3.2 Database Management:	10
3.3.3 Developing the Flask Environment:.....	10
3.3.4 Data Visualization:.....	10
3.3.5 Population Projection:	11
3.3.6 Dashboard Development:	11
3.4 Data Analysis.....	11
4. Code Integration.....	12

4.1	Flask Application (app.py)	12
4.2	HTML Template (index.html).....	15
4.3	CSS File (index.css)	17
4.4	JavaScript File (main.js).....	20
5.	Analysis and Data Interpretation.....	26
5.1	The population data	26
5.2	Population Simulation	28
6.	Challenges, Limitations and Roadmap for Future Development.....	31
7.	Conclusion	32
8.	References.....	33
9.	Appendix	34

1. Introduction

1.1 Background Information

Population growth is a dynamic and multifaceted phenomenon that significantly impacts various aspects of society, including urban planning, economic development, and resource management. Understanding how populations change over time is crucial for governments, policymakers, and researchers to make informed decisions that ensure sustainable development and efficient allocation of resources.

Baden-Württemberg, a state in southwestern Germany, has undergone substantial demographic changes over the past decades. These changes have been influenced by factors such as birth rates, death rates, migration patterns, and economic conditions. Accurate and accessible population data is essential for tracking these trends and making projections about future population growth.

With the proliferation of digital technologies, online mapping and data visualization tools have become indispensable for demographic analysis. These tools enable users to interact with data in real-time, offering a more intuitive and comprehensive understanding of population dynamics. In this project, we leverage cutting-edge web technologies—Flask for the backend, PostgreSQL for database management, and MapLibre for interactive mapping—to create a robust platform for visualizing and analyzing population growth in Baden-Württemberg.

1.2 Objectives

- To develop a dashboard interface that integrates open data on population growth from the Landesamt Baden-Württemberg.
- To create interactive maps using Flask and Maplibre to visualize population distribution and trends.
- To design presentable graphs and charts that provide insights into population growth patterns over time.

1.3 Significance of the Study

The ability to visualize population data interactively and accurately is invaluable for a variety of stakeholders. Policymakers can use these insights to plan for future infrastructure needs, such as schools, hospitals, and transportation systems. Urban planners can better understand

where to focus development efforts, and social scientists can gain deeper insights into demographic shifts and their underlying causes.

Moreover, projecting future population trends is crucial for anticipating potential challenges and opportunities. For instance, an aging population may require more healthcare services, while areas with rapidly growing populations might need expanded educational facilities and housing developments. By providing a platform that integrates historical data with future projections, this project aims to contribute to more informed and proactive planning.

1.4 Project Relevance

This project is particularly relevant in the context of smart city initiatives and the increasing reliance on data-driven decision-making. By making demographic data accessible and understandable, we empower communities and decision-makers to respond more effectively to the changing needs of their populations. Additionally, the use of open data and open-source tools aligns with the principles of transparency and collaboration, fostering a more inclusive approach to data analysis and urban planning.

1.5 Innovation and Technological Integration

Our approach combines traditional demographic analysis methods with modern web technologies to create an innovative solution for population data visualization. Flask, a lightweight web framework, allows us to build a scalable and responsive backend. PostgreSQL, a powerful relational database, ensures efficient data storage and retrieval. MapLibre provides the tools needed to create highly interactive and user-friendly maps, making complex data more accessible and engaging.

By integrating these technologies, we offer a novel way to explore and understand population dynamics, bridging the gap between raw data and actionable insights. This project not only demonstrates the practical applications of these technologies but also serves as a model for future efforts in data visualization and demographic analysis.

2. Literature Review

Population growth analysis and visualization are well-explored fields, with numerous studies emphasizing the importance of accurate and interactive tools for demographic analysis.

2.1 Open Data Utilization

2.1.1 Open Data for Research

The availability of open data from government sources like the Landesamt Baden-Württemberg enables researchers to conduct detailed demographic studies without significant resource constraints (Kitchin, 2014). Open data policies encourage the sharing of information, promoting collaboration and innovation.

2.1.2 Case Studies in Open Data

Various studies have demonstrated the value of open data in enhancing public services and informed decision-making. For example, the use of open demographic data in urban planning has led to more efficient allocation of resources and better service delivery (Janssen et al., 2012).

2.2 Population Growth Analysis

2.2.1 Demographic Analysis Techniques

Traditional methods of population projection, such as the cohort-component method, have been widely used. This method accounts for births, deaths, and migration to forecast future population changes (Preston et al., 2001).

2.2.2 GIS and Population Studies

Geographic Information Systems (GIS) have revolutionized demographic studies by providing spatial analysis tools that can visualize population distribution and trends (Goodchild, 2007).

2.3 Application of Population Projection Techniques

Projecting future population trends involves sophisticated models that account for various demographic factors. These projections are crucial for planning and policy-making.

2.3.1 Exponential Population Projection

The exponential growth method is commonly used for projecting future populations, especially when historical data suggests a constant rate of growth. The exponential growth formula is:

$$x(t) = x_0 \times (1 + r)^t$$

Where, $x(t)$: Final population after time t

x_0 : Initial population

r : Rate of growth

t : Total time (number of years)

- **Application of the Formula:** To apply this formula, accurate estimates of the current population and the growth rate are required. These parameters are often derived from historical data analysis and demographic studies (Smith, Tayman, & Swanson, 2013).

2.3.2 Innovations in Population Projection

Recent advancements in computational methods and data availability have improved the accuracy and usability of population projections. Machine learning and other data-driven approaches are increasingly being integrated into traditional projection models to enhance their predictive power (Bongaarts & Bulatao, 2000).

2.4 Technologies for Data Management and Visualization

2.4.1 Flask

Flask is a lightweight WSGI web application framework in Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. Flask's simplicity and flexibility have made it a popular choice for developing RESTful APIs and web applications (Grinberg, 2018).

- **Advantages in Web Development:** Flask's modular design allows developers to choose the components they need, making it adaptable to a wide range of projects. Its extensive documentation and supportive community contribute to its usability and robustness.
- **Use in Demographic Studies:** In demographic studies, Flask can be used to develop web applications that interact with databases to retrieve and display population data

dynamically. For example, Flask can serve as the backend for a population dashboard, handling API requests and processing data for visualization.

2.4.2 PostgreSQL

PostgreSQL is a powerful, open-source relational database system known for its reliability, feature robustness, and performance. It supports advanced data types and performance optimization features, making it ideal for complex data queries and large datasets (Momjian, 2001).

- **Features and Capabilities:** PostgreSQL supports a wide range of data types and advanced querying capabilities, including full-text search, spatial data with PostGIS, and JSON support. Its robustness in handling large volumes of data makes it suitable for storing and querying demographic data.
- **Application in Data Management:** In the context of population studies, PostgreSQL can store historical and current population data, allowing for complex queries and analysis. It can be integrated with web applications through APIs, enabling real-time data retrieval and updates.

2.4.3 pgAdmin

pgAdmin is a comprehensive open-source management tool for PostgreSQL, offering a graphical interface for database management and administration. It simplifies database interactions, allowing users to create, modify, and query databases with ease (pgAdmin Development Team, 2020).

- **Features and Benefits:** pgAdmin provides a user-friendly interface for database administration, including tools for designing, managing, and monitoring databases. It supports SQL queries, data import/export, and backup/restore operations.
- **Role in Database Management:** For demographic data management, pgAdmin can be used to import population datasets, execute queries for data analysis, and manage database schemas. Its visualization tools help in understanding data structures and relationships, facilitating efficient data handling and analysis.

2.5 Online Mapping and Visualization Tools

2.5.1 MapLibre

MapLibre is an open-source mapping library that originated as a community-driven fork of Mapbox GL JS. It is designed for creating highly interactive and customizable vector maps that are performant on both desktop and mobile devices (MapLibre, 2020). MapLibre's ability to handle large datasets efficiently makes it an ideal tool for visualizing population data, where the density and distribution of populations can vary widely across geographic regions.

- **Customization and Interactivity:** MapLibre provides extensive customization options, allowing developers to create maps that precisely meet their needs. Features such as custom styles, dynamic data layers, and interactive controls enhance the user experience and facilitate deeper data exploration.
- **Performance and Scalability:** One of the key strengths of MapLibre is its performance, particularly when handling large datasets and complex visualizations. This scalability ensures that maps remain responsive and interactive even when visualizing extensive demographic data.
- **Use Cases in Demographic Studies:** Several studies and projects have utilized MapLibre to visualize demographic data effectively. For instance, interactive maps created with MapLibre have been used to display real-time population density, migration patterns, and urban growth trends, providing valuable insights for researchers and policymakers (Smith, 2021).
- **Web-based Dashboards:** Dashboards integrate various data visualization techniques, including charts and maps, to present data comprehensively. They are essential for real-time data analysis and decision-making (Few, 2006).

3. Methodology

3.1 Study Design

This study involves the development of a web-based dashboard to visualize and project population growth. The project integrates various technologies and data sources to create a comprehensive tool for demographic analysis.

3.2 Materials

- **Software and Tools:** Flask, MapLibre, PostgreSQL, Excel.
- **Data Source:** Population data from the Landesamt Baden-Württemberg, provided as CSV files.

3.3 Procedure

3.3.1 Data Collection and Preparation:

- ✓ Download population data from the Landesamt Baden-Württemberg.
- ✓ Clean and preprocess the data to correct any inconsistencies.
- ✓ Import the cleaned data into PostgreSQL using pgAdmin.

3.3.2 Database Management:

- ✓ Set up a PostgreSQL server using pgAdmin.
- ✓ Store the population data in a structured format to facilitate easy access and querying.

3.3.3 Developing the Flask Environment:

- ✓ Set up a Flask environment to create the backend of the web application.
- ✓ Develop APIs to fetch data from the PostgreSQL database.

3.3.4 Data Visualization:

- ✓ Use MapLibre to create interactive maps that display population distribution.
- ✓ Implement charts and graphs to visualize population growth trends over time.

3.3.5 Population Projection:

- ✓ Apply the Exponential population projection formula to estimate population for 2025, 2030, 2035, and 2040.
- ✓ Integrate the projected data into the dashboard for visualization.

3.3.6 Dashboard Development:

- ✓ Design and develop a user-friendly dashboard interface.
- ✓ Ensure the dashboard displays interactive maps, charts, and population projections.

3.4 Data Analysis

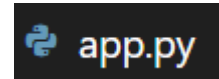
- **Spatial Analysis:** Visualize population distribution and trends using interactive maps.
- **Temporal Analysis:** Analyze population growth over time through charts and graphs.
- **Projection Analysis:** Use the componential population projection formula to estimate future population trends and visualize these projections

4. Code Integration

In this section, we will explain the details of the implementation of the dashboard. This dashboard uses Flask to serve a web application that includes a map and charts to analyze population growth. The main files involved are app.py, index.html, index.css and main.js.

4.1 Flask Application (app.py)

The app.py file is the backbone of our Flask application, providing routes and endpoints for serving HTML templates and data in JSON format.



The Flask application begins with importing the necessary modules: Flask for creating the web application, psycopg2 for connecting to the PostgreSQL database, and geojson for handling geographical data. The application instance is created with `app = Flask(__name__)`, and the database connection is established using `psycopg2.connect()`, with the credentials provided to connect to a PostgreSQL database named "GSS". A cursor `cur` is also created to execute SQL queries.

```
from flask import Flask, render_template, jsonify, g
import psycopg2
import geojson

app = Flask(__name__)

✓ connection = psycopg2.connect(
    user="postgres",
    password="postgres",
    host="localhost",
    port="5432",
    database="GSS"
)

print('Connection Success')
cur = connection.cursor()
```

Figure 1 Importing modules and connecting to the database for Flask application

Several routes were defined to handle different parts of our application. HTTP routes are endpoints defined on a server that respond to HTTP requests. When a user makes a request to one of these routes by searching the route in the browser, the server processes the request and sends back an appropriate response which is all the information that defined in this route. Here for example the `/map.html` route renders the main map page using the `index.html` template which is going to visualize the whole map.

```
@app.route("/map.html")
def get_map():
    return render_template("index.html")
```

Figure 2 Defining routes for responding to the user requests

One crucial route in our HTML file is `geojson`, which fetches geographical data from the database, converts it to GeoJSON format, and returns it as a JSON response. JSON is a data interchange format. It means a standard way of encoding data so that it can be easily shared and understood between different systems, applications or components. It is mostly use in web applications that there is a server and client and they need a common language to make connection. GeoJSON also uses JSON for encoding spatial data structures, such as points, lines, polygons, and other geometrical shapes. It is often used in web mapping and GIS applications to represent geographical features and their properties.

In `geojson` route data is fetched from our database in PostgreSQL by defining a SQL query ("SELECT id, name, schlüssel, ST_AsGeoJSON(geom) AS geometry FROM data;") that selects the necessary fields and converts the geometry field to GeoJSON using `ST_AsGeoJSON()`. The results are processed into a list of GeoJSON features, which are then combined into a GeoJSON feature collection and returned.

```
@app.route('/geojson')
def get_geojson_data():
    cur.execute(
        """SELECT id, name, schlüssel, ST_AsGeoJSON(geom) AS geometry FROM data;""")
    multipolygon_geojson = cur.fetchall()

    features = []
    index = 1
    for row in multipolygon_geojson:
        geometry = geojson.loads(row[3])
        feature = {
            "type": "Feature",
            "geometry": geometry,
            "id": index,
            "properties": {
                'id': row[0],
                'name': row[1],
                'schlüssel': row[2]
            }
        }
        features.append(feature)
        index +=1

    feature_collection = {
        "type": "FeatureCollection",
        "features": features
    }

    return jsonify(feature_collection)
```

Figure 3 Fetching data from database using SQL query

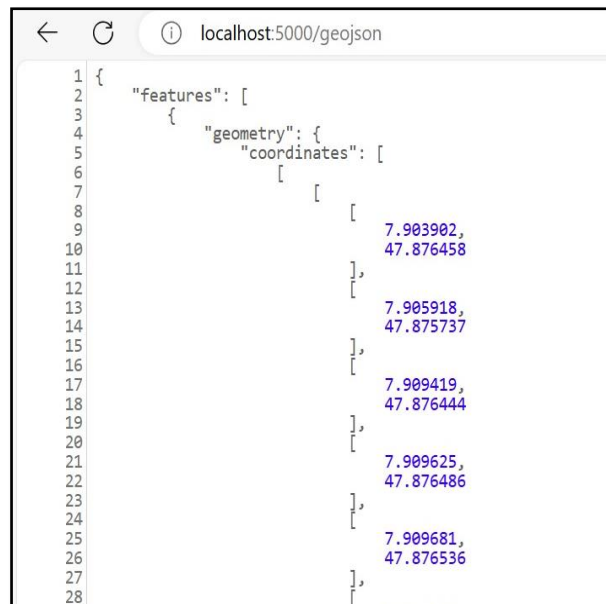


Figure 4 The geojson route in the webpage

Additional routes are defined to fetch data for all of our location of interests which are the cities Esslingen, Karlsruhe and Ludwigsburg. For example, the population/espop route executes a SQL query to fetch population data for Esslingen ("select distinct on (year) 'Esslingen Population' as title, year, poppersqkm from espop order by year;") then processes the results and returns it as a JSON response. Then we do the same procedure for the other cities that we have. For the simulation part of the map we also need to the same procedure separately for all the cities.

```
#Chat Creation
@app.route('/population/espop')
def get_espop_data():
    cur.execute("select distinct on (year) 'Esslingen Population' as title, year, poppersqkm
    data = cur.fetchall()

    features = []

    for row in data:
        feature = {
            "title": row[0],
            "year": row[1],
            "population": row[2],
        }


        features.append(feature)

    return jsonify(features)
```

Figure 5 Defining route for fetching the data related to the cities



Figure 6 The population route in the webpage

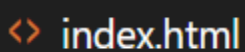
At the end we defined the option for running the application in debug mode with `app.run(debug=True)`, which is helpful for development as it provides detailed error messages in the inspect option  Inspect of the webpage and automatically reloads the application when changes are made.

```
if __name__ == '__main__':
    app.run(debug=True)
```

Figure 7 Assigning the debug mode for the code to show the errors

4.2 HTML Template (index.html)

The index.html file is the frontend template for our application. It includes the structure and styling layout for the dashboards including map and control elements. It starts with the metadata and links to external stylesheets and scripts.



In this file we use Maplibre GL which is a JavaScript library for interactive and customizable vector maps on the web. It uses Web Graphics library which is a JavaScript API that allows web browsers to render interactive 2D and 3D graphics. So it helps to render maps and provides functionalities like markers, popups, layers, interactivity with geographic data. We use also the Chart.js which is a popular JavaScript library for creating a variety of charts like line, bar and pie and it is easy to integrate and customize.

```

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Population Data Portal</title>
  <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no" />

  <!-- Maplibre GL -->
  <link href="https://unpkg.com/maplibre-gl@2.4.0/dist/maplibre-gl.css" rel="stylesheet" />

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <link rel="stylesheet"
    href="https://fonts.googleapis.com/css2?family=IBM+Plex+Sans:wght@200;300;400;500;600;700&display=swap">
  <link rel="stylesheet" href="{ { url_for('static', filename='css/index.css') } }">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>University Logo</title>
</head>

```

Figure 8 The metadata definition and connection to external stylesheets and scripts

The body part of the this file divide the webpage into different sections and containers using div elements and it also includes buttons to organize the content.

One of this elements is title section which includes the name of the project and the producers. One container is defined for the map, where the interactive map will be rendered. Also a control panel contains buttons are defined which include the radio buttons for selecting specific datasets for each three cities Esslingen, Karlsruhe and Ludwigsburg; also a button for submitting to show the related data.

```

<body>
  <div class="titleDiv">
    <h1>Online Map for Analyzing Population Growth of Baden-Württemberg</h1>
    <h2>Prepared by: Md. Raisul Islam, Md Ashifur Rahaman & Fatemeh Rafiei</h2>
  </div>
  <ul>
    {% for row in data %}
    <li>{{ row }}</li>
    {% endfor %}
  </ul>
  <div id="map"></div>
  <div class="credit"></div>

  <div class="control-container">
    <button id="populationBtn" class="active">Population</button>
    <button id="simulationBtn">Simulation</button>
    <div id="populationOptions" class="options">
      <label><input type="radio" name="population" id="espop" value="espop"> Esslingen Pop</label>
      <label><input type="radio" name="population" id="karlpop" value="karlpop"> Karlsruhe Pop</label>
      <label><input type="radio" name="population" id="ludpop" value="ludpop"> Ludwigsburg Pop</label>
    </div>
    <div id="simulationOptions" class="options hidden">
      <label><input type="radio" name="simulation" id="esssim" value="esspop"> Esslingen Sim</label>
      <label><input type="radio" name="simulation" id="karlsim" value="karlpop"> Karlsruhe Sim</label>
      <label><input type="radio" name="simulation" id="ludsim" value="ludpop"> Ludwigsburg Sim</label>
    </div>
    <button id="submitBtn">Submit</button>
  </div>

```

Figure 9 Defining div elements for dividing the webpage into different containers

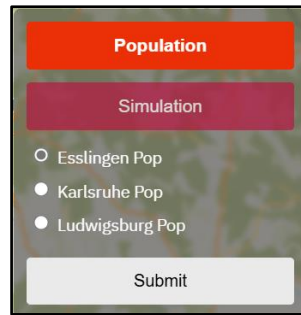


Figure 10 The visualization of the control container in the webpage

There is also another container for the chart which is hidden in the first step and when the user selects and submits a dataset it will show up. This container includes also a close button and a canvas element for rendering the chart.

```
<div id="chartContainer" class="hidden">
  <button id="closeChartBtn">Close</button>
  <canvas id="myChart"></canvas>
</div>
```

Figure 11 Defining the chart container including close button

4.3 CSS File (index.css)

The CSS file provides styling for the HTML elements. In this file we can add features to the map to have a user-friendly view.

index.css

Basic styles such as font and size of the webpage and margins can apply to the body. One other element is the title which is positioned at the top of the page with a transparent background because we don't want to cover some part of the map with title and it contains h1 and h2 elements which is the style of the title text, the text itself defined in the title div of index.html. The map container is set to cover the entire viewport, and other features like control container and charts will be cover some part of that.

```
body {
  margin: 0;
  padding: 0;
  overflow: hidden;
  font-family: 'IBM Plex Sans', sans-serif;
}

#map {
  position: absolute;
  width: 100%;
  height: 100vh;
  background-color: #eaeaea; /* Placeholder for the map */
  z-index: 1;
  top: 0;
}
```

Figure 12 Styling of the map in index.css file

```

✓ .titleDiv {
  position: absolute;
  top: 0;
  align-items: center;
  justify-content: center;
  padding: 5px;
  backdrop-filter: blur(5px);
  z-index: 10;
  display: flex;
  flex-direction: column;
  width: 100%;
}

```

Figure 13 styling of the title in the webpage

The control container is positioned in the top right side of the webpage to make it easily accessible, and it includes the population, simulation and submit button with specific styles for each including the hover feature. The item labels are also styled for easy selection.

```

✓ .control-container {
  position: absolute;
  color: #fff;
  top: 30px;
  right: 10px;
  background: rgba(0, 0, 0, 0.4);
  padding: 10px;
  border-radius: 5px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  z-index: 1000;
  backdrop-filter: blur(2px);
  width: 13rem;
}

✓ #submitBtn {
  display: block;
  width: 100%;
  margin: 10px 0 0 0;
  padding: 10px;
  background-color: #eaeaea;
  color: #000;
  border: 1px solid transparent;
  border-radius: 3px;
}

✓ #submitBtn:hover {
  background-color: #78e08f;
  cursor: pointer;
  transition: 0.3s linear;
}

```

Figure 14 styling of the map container in the webpage

The option class defined to organize child elements population, simulation into a vertical column and elements with the hidden class will not display it means that when one of them selected the other one will be in the hidden format.

```

✓ .options {
  display: flex;
  flex-direction: column;
}

✓ .options label {
  margin-bottom: 7px;
  font-size: 13px;
}

✓ .hidden {
  display: none;
}

```

Figure 15 Styling the list of items in a vertical column

The chart container is also positioned at the bottom-left, and styled with transparent background. The close chart button provides a clickable close button in the chart container. The credit div also fixed at the bottom right, displaying the logo of the HFT Stuttgart, and styled to have shadow.

```

#chartContainer {
  position: absolute;
  width: 500px;
  height: 250px;
  left: 10px;
  bottom: 0;
  background: rgba(255, 255, 255, 0.9);
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  z-index: 1000; /* Ensure chart is on top */
  display: flex;
  flex-direction: column;
  align-items: center;
}

#closeChartBtn {
  background-color: #ff4b5c;
  color: #fff;
  border: none;
  border-radius: 3px;
  cursor: pointer;
  padding: 5px 10px;
  position: absolute;
  bottom: 10px;
  right: 10px;
  font-size: 0.8em;
}

.credit {
  width: 100px;
  height: 100px;
  overflow: hidden;
  position: fixed;
  z-index: 99999;
  background: url(/Images/hft.jpg) no-repeat;
  background-size: 100px;
  bottom: 40px;
  right: 10px;
  box-shadow: 0 0 10px 1px #000;
}

```

Figure 16 Styling the chart container

4.4 JavaScript File (main.js)

The JavaScript file is responsible for handling user interactions, updating the UI, and communicating with the server for interacting with the map and fetching data from the server.

JS main.js

In this file a map from the Maplibre GL library is created in a specific location which would be the center of the map and with the zoom level value which is defined. The capability of zooming and rotating also added to the map. GeoJSON data fetched from the geojson endpoint (which defined in the app.py file) using Maplibre and added as a source and layer to the map for visualization.

```
var maplibreMap = new maplibregl.Map({
  container: 'map', // container ID
  style: {
    'version': 8,
    'sources': {
      'osm-tiles': {
        'type': 'raster',
        'tiles': [
          'https://a.tile.openstreetmap.org/{z}/{x}/{y}.png',
          'https://b.tile.openstreetmap.org/{z}/{x}/{y}.png',
          'https://c.tile.openstreetmap.org/{z}/{x}/{y}.png'
        ],
        'tileSize': 256,
        'attribution': '© OpenStreetMap contributors'
      }
    },
    'layers': [
      {
        'id': 'osm-tiles',
        'type': 'raster',
        'source': 'osm-tiles',
        'minzoom': 0,
        'maxzoom': 19
      }
    ]
  },
  center: [9.1544, 48.6361], // starting position [lng, lat]
  zoom: 8 // starting zoom
});

// Add zoom and rotation controls to the map.
maplibreMap.addControl(new maplibregl.NavigationControl());

var geojsonUrl = "http://localhost:5000/geojson";

maplibreMap.on('load', async function () {
  maplibreMap.addSource('geojson_data', {
    'type': 'geojson',
    'data': geojsonUrl
  });
});
```

Figure 17 Creating map using Maplibre library

The map features also should be styled. It is done by defining two layers: One for the filled areas of the features with colors. This procedure is based on their interaction option which can be hover or click and changing the opacity when hovered. The second layer is for defining the specific features for outlines of the area polygons.

```

maplibreMap.addLayer({
  'id': 'geojson_layer',
  'type': 'fill',
  'source': 'geojson_data',
  'layout': {},
  'paint': {
    'fill-color': [
      'case',
      ['boolean', ['feature-state', 'clicked'], false], '#00FF00', // clicked color
      ['boolean', ['feature-state', 'hover'], false], '#FFFF00', // hover color
      '#627BC1' // default color
    ],
    'fill-opacity': ['case', ['boolean', ['feature-state', 'hover'], false], 0.5, 0.8]
  }
});

maplibreMap.addLayer({
  'id': 'geojson_layer_outline',
  'type': 'line',
  'source': 'geojson_data',
  'layout': {},
  'paint': {
    'line-color': '#000000',
    'line-width': 0.5
  }
});

```

Figure 18 Add layers for styling the geojson data

Event listener is a function in JavaScript that when user clicks on the webpage executes a specific piece of code as a response to the action which helps for interactivity of the map.

Using this function, in the following code, when the user clicks on a feature in the map which includes the GeoJSON data, it first checks if this feature was clicked before and then updates the state of the current clicked feature to indicate the properties of the clicked feature which is the name of the city, as a popup.

```

maplibreMap.on('click', 'geojson_layer', function (e) {
  if (clickedFeatureId) {
    maplibreMap.setFeatureState(
      { source: 'geojson_data', id: clickedFeatureId },
      { clicked: false }
    );

    clickedFeatureId = e.features[0].id;

    maplibreMap.setFeatureState(
      { source: 'geojson_data', id: clickedFeatureId },
      { clicked: true }
    );

    var coordinates = e.lngLat;
    var properties = e.features[0].properties;

    // Log the properties to the console
    console.log(properties);

    // Create a popup
    var popupContent = '<b>' + properties.name + '</b>';
    new maplibregl.Popup()
      .setLngLat(coordinates)
      .setHTML(popupContent)
      .addTo(maplibreMap);
  }
});

```

Figure 19 Defining event listener for visualization of the popup



Figure 20 Popup visualization in the webpage

This function also attached to the buttons for switching between population and simulation data. When the user clicks on the population button, the population options are shown, and the simulation options are getting hide.

```
document.addEventListener('DOMContentLoaded', function () {
  const populationBtn = document.getElementById('populationBtn');
  const simulationBtn = document.getElementById('simulationBtn');
  const populationOptions = document.getElementById('populationOptions');
  const simulationOptions = document.getElementById('simulationOptions');
  const submitBtn = document.getElementById('submitBtn');
  const chartContainer = document.getElementById('chartContainer');
  const closeChartBtn = document.getElementById('closeChartBtn');
  const ctx = document.getElementById('myChart').getContext('2d');
  let myChart;

  populationBtn.addEventListener('click', function () {
    populationBtn.classList.add('active');
    simulationBtn.classList.remove('active');
    populationOptions.classList.remove('hidden');
    simulationOptions.classList.add('hidden');
  });
});
```

Figure 21 Defining event listener for switching between radio buttons based on the clicks

Another button which is defined here is submit button that when the user selects a dataset and clicks the submit button, the application fetch the selected data from the server then processed it and then stores the url, defined route in the app.py file, for that dataset.

```

submitBtn.addEventListener('click', function () {
  let apiUrl = '';
  let title = '';
  if (populationBtn.classList.contains('active')) {
    if (document.getElementById('espop').checked) {
      apiUrl = 'http://localhost:5000/population/espop';
      title = 'Esslingen Population';
    } else if (document.getElementById('karlpop').checked) {
      apiUrl = 'http://localhost:5000/population/karlpop';
      title = 'Karlsruhe Population';
    } else if (document.getElementById('ludpop').checked) {
      apiUrl = 'http://localhost:5000/population/ludpop';
      title = 'Ludwigsburg Population';
    }
  } else if (simulationBtn.classList.contains('active')) {
    if (document.getElementById('essim').checked) {
      apiUrl = 'http://localhost:5000/simulation/espop';
      title = 'Esslingen Simulation';
    } else if (document.getElementById('karlsim').checked) {
      apiUrl = 'http://localhost:5000/simulation/karlpop';
      title = 'Karlsruhe Simulation';
    } else if (document.getElementById('ludsim').checked) {
      apiUrl = 'http://localhost:5000/simulation/ludpop';
      title = 'Ludwigsburg Simulation';
    }
  }
}

```

Figure 22 Fetching the related route to the data related to the selected city

The default view which we want to have is just showing map view without any chart container. In this situation the hide function can help to make the chart invisible.

```

// Hide chart container on page load
$('#chartContainer').hide();

// Show chart container on submit button click
$('#submitBtn').click(function () {
  // Your existing submit button logic goes here

  // Show the chart container
  $('#chartContainer').show();
});

```

Figure 23 defining the hide and show function for the chart container

The show function renders the chart using the stored url of the route that we defined in the app.py file. It extracts the necessary data from the response, such as labels years and population values, and configures the chart with these data points. The chart container is displayed, and a line chart is created with the fetched data.

```

if (apiUrl) {
  fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
      console.log('Success:', data);
      // Prepare data for the chart
      const labels = data.map(item => (item.year));
      const values = data.map(item => item.population);
    });
}

```

Figure 24 Preparing data from the url of the selected city

In case user wants to visualize another chart for the another data based on selection, the following code allows the visualization of a new chart based on user selection, ensuring that any previously existing chart is destroyed before creating a new one.

```

// Destroy previous chart instance if it exists
if (myChart) {
  myChart.destroy();
}

// Create a new chart
myChart = new Chart(ctx, {
  type: 'bar',
  data: {
    labels: labels,
    datasets: [{
      label: 'Population per Year',
      data: values,
      borderColor: '#e056fd',
      borderWidth: 1,
      fill: false
    }]
  },
  options: {
    responsive: true,
    scales: {
      x: {
        type: 'linear',
        position: 'bottom',
        title: {
          display: true,
          text: 'Year'
        }
      },
      y: {
        beginAtZero: true, // Begin y-axis at 0
        title: {
          display: true,
          text: 'Population(thousand)'
        }
      }
    },
    plugins: {
      title: {
        display: true,
      }
    }
  }
});

```

Figure 25 Switching between the charts based on the selection

For return to the map view (the situation that there is no other window selected from the map), the close button defined in which within the chart container after clicking the chart will be hidden.

```

$('#closeChartBtn').click(function () {
  // Hide the chart container
  $('#chartContainer').hide();
});

```

Figure 26 Defining close button for disappearing the charts

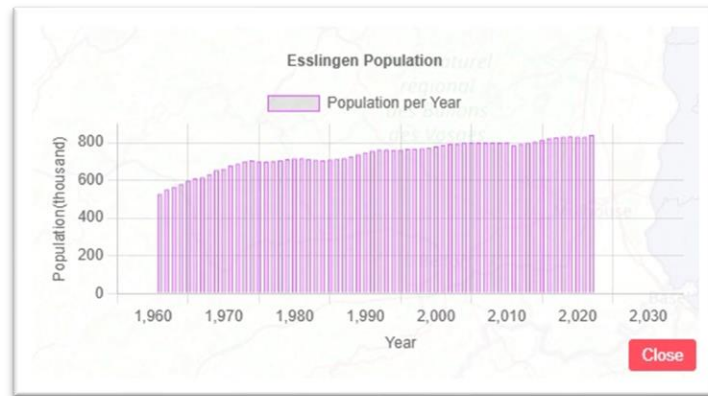


Figure 27 The chart visualization in the webpage

Final look of the Web Map

Overall, these 4 components work together to create an interactive dashboard for exploring geographical and population data, providing both a visual map and detailed charts for analysis.

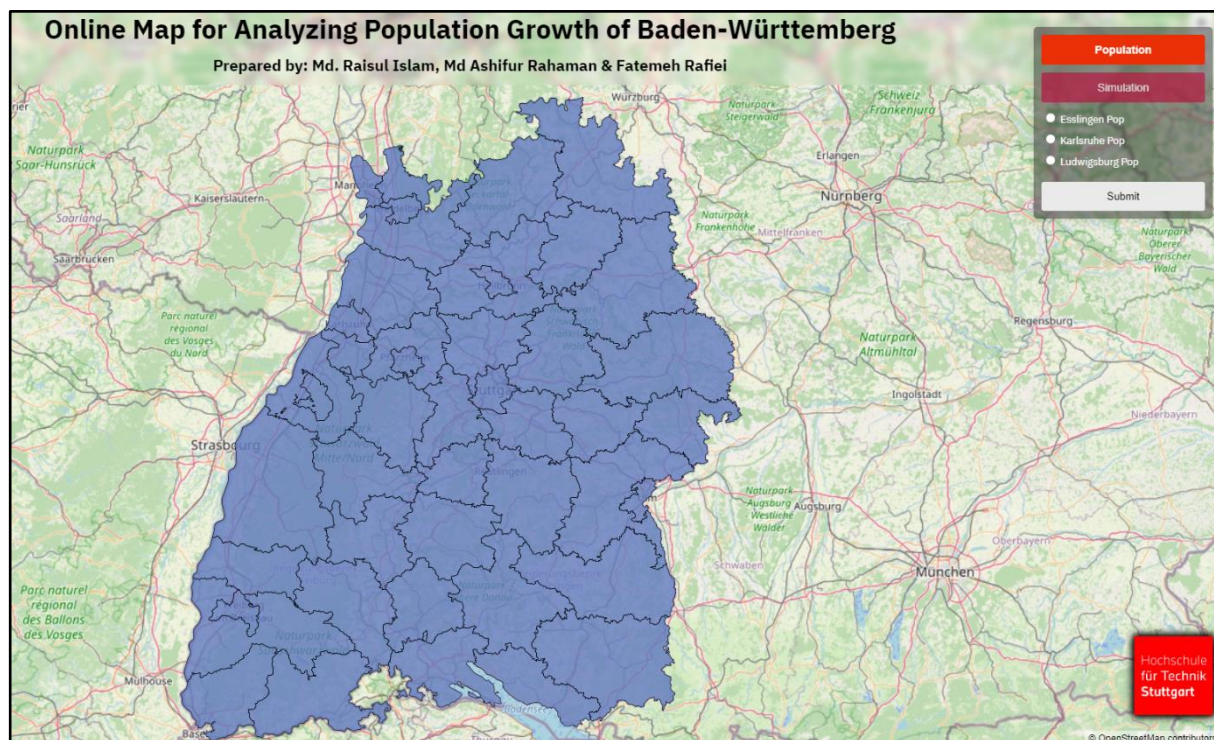


Figure 28 The dashboard visualization

5. Analysis and Data Interpretation

5.1 The population data

Esslingen, Ludwigsburg and Karlsruhe districts of Baden-Württemberg are used for the project from year 1961 to 2022. After selecting the population option and Esslingen district, a chart appears with the information (Figure 29).

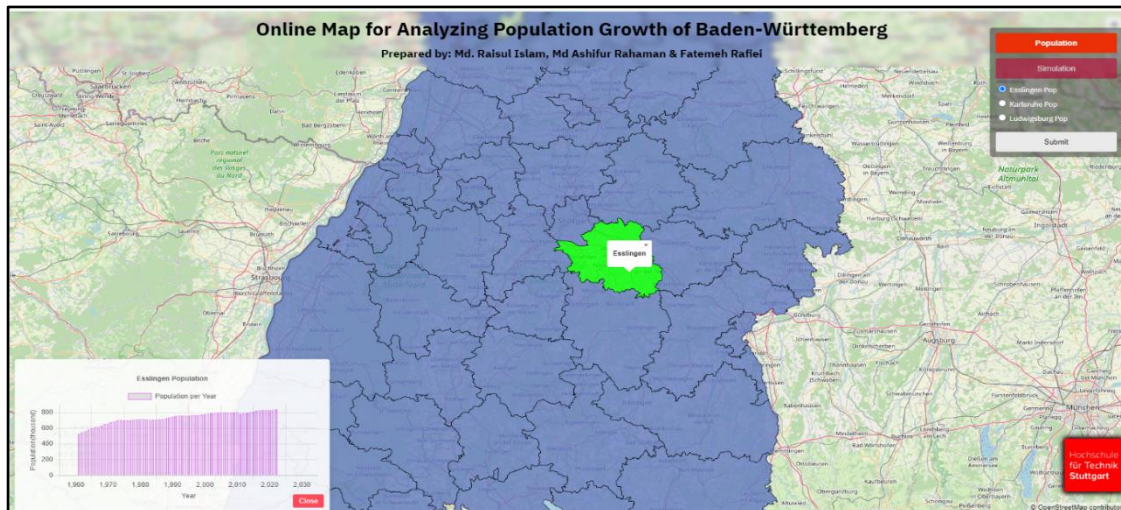


Figure 29: Population for Esslingen (from Year 1961 to 2022)

The fluctuations in the population of Esslingen am Neckar from 1980 to 2022 can be attributed to several factors. Between 1980 and 2010, the population declined due to economic changes, such as industrial downturns and job losses, prompting people to move elsewhere for better opportunities. Additionally, urbanization trends likely drew residents to larger cities like Stuttgart, offering more amenities and job prospects. An aging population with lower birth rates further contributed to the decline. However, from 2010 to 2020, the population began to rise again, possibly due to economic recovery, improved infrastructure, and an influx of immigrants, making Esslingen more attractive. Post-2020, the population saw another dip, potentially influenced by the COVID-19 pandemic's economic impact, increased mortality rates, and continued urbanization trends.

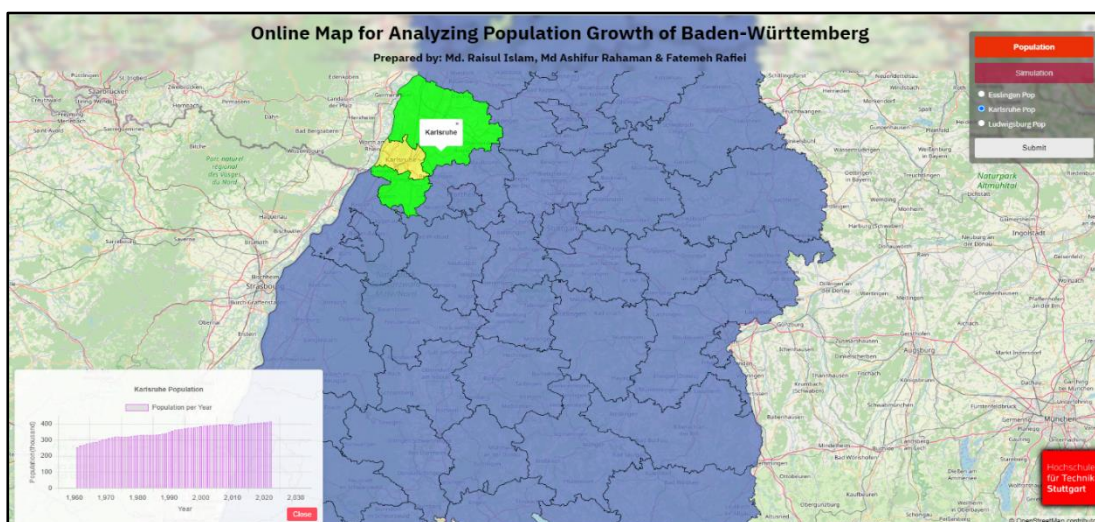


Figure 30: Population for Karlsruhe (from Year 1961 to 2022)

The chart of Karlsruhe (Figure 30) has shown a consistent upward trend from 1961 to 2022. This growth can be attributed to several factors. Between 1961 and 2000, the city's economic development, driven by its industrial and technological sectors, attracted a growing workforce. Additionally, Karlsruhe's status as an educational hub, home to the renowned Karlsruhe Institute of Technology (KIT), drew students and academics. Post-2000, the population growth accelerated, reflecting further economic prosperity, improved infrastructure, and the city's appeal as a desirable place to live and work.

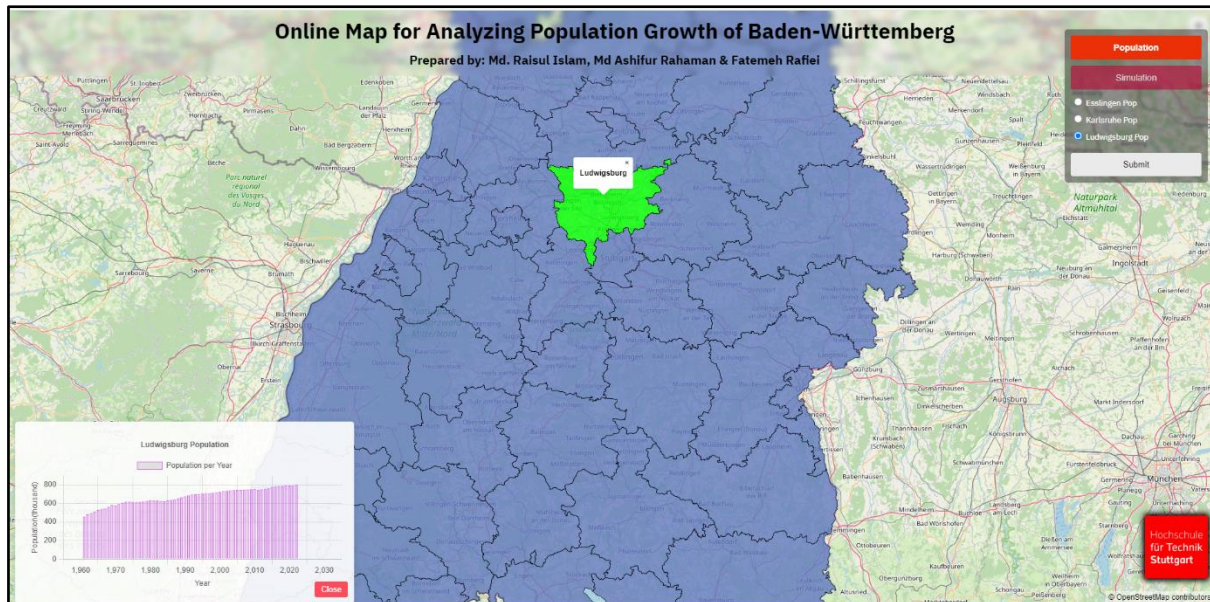


Figure 31: Population for Ludwigsburg (from Year 1961 to 2022)

The population of Ludwigsburg (Figure 31) has consistently increased from 1961 to 2022, reflecting its evolution as a prominent urban center in the Stuttgart metropolitan area. The city saw continuous growth due to economic expansion in manufacturing and services, which attracted more residents seeking employment opportunities. Ludwigsburg's cultural heritage, educational institutions, and proximity to Stuttgart further enhanced its appeal as a residential destination. By 2000, the population had risen significantly, and ongoing urban development and infrastructure improvements continued to bolster its attractiveness.

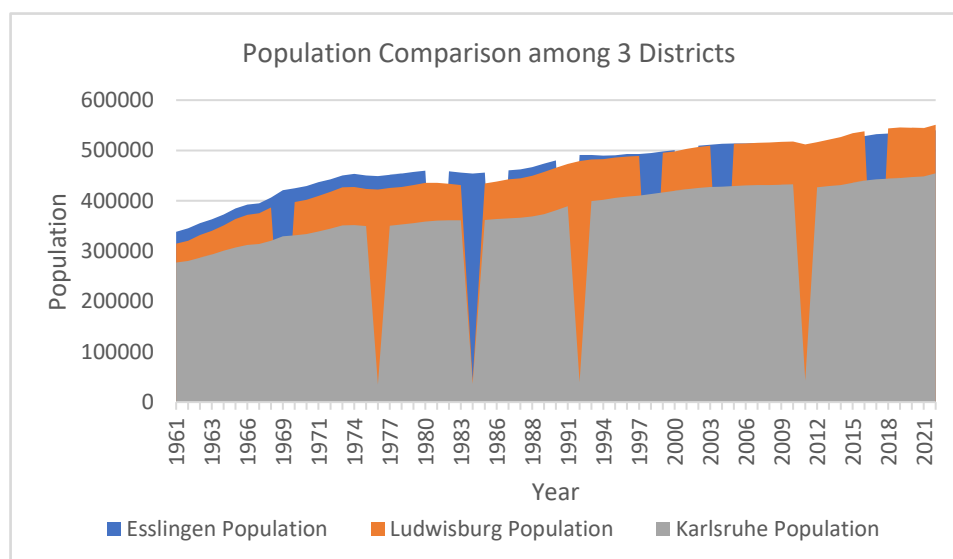


Figure 32: Population Comparison among the 3 Districts

From 1960 to 2022, Esslingen, Karlsruhe, and Ludwigsburg have each followed distinct paths in population growth (Figure 32). Esslingen has seen gradual increases, buoyed by its industrial base and scenic location along the Neckar River. Karlsruhe, known for its innovation and cultural richness, has experienced more pronounced growth, driven by its thriving technology and academic sectors. Ludwigsburg, nestled in the Stuttgart metropolitan area, has exhibited the most significant expansion, benefiting from economic development in manufacturing and services, coupled with its cultural attractions and proximity to Stuttgart. Each city's demographic trajectory reflects its unique blend of economic opportunities, cultural amenities, and regional significance over the past six decades.

5.2 Population Simulation

Another goal of the project is to simulate the population for the 3 districts for the coming years. In the web application the simulation for population is done for up to Year 2040. By clicking the simulation button and selecting the district, it will show the simulation for the district in a

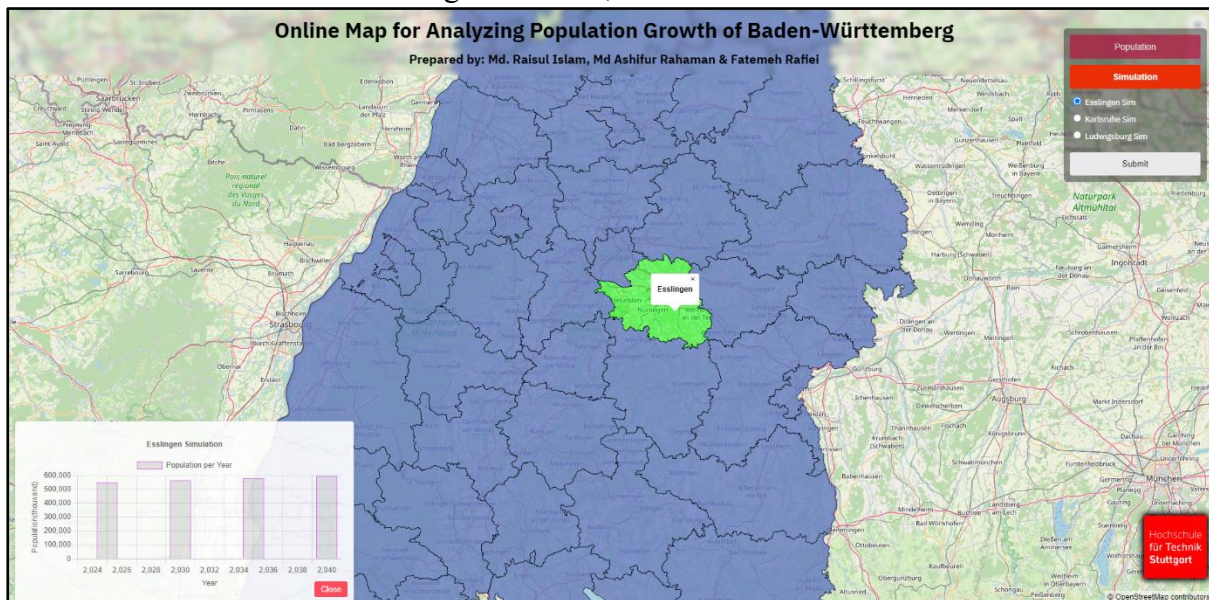


Figure 33: Population Simulation for Esslingen (from year 2022 to 2040)

The population projection for Esslingen (Figure 33) indicates a consistent and steady growth trajectory based on an exponential growth formula. Starting from 549,189 in 2025 and projected to reach 596,284 by 2040, these figures underscore Esslingen's ongoing appeal as a residential and economic center within the Stuttgart metropolitan region. The city's historical significance, coupled with its industrial heritage and scenic location along the Neckar River, continues to attract new residents and businesses alike. This population expansion reflects sustained economic opportunities in manufacturing, services, and commerce, as well as the city's role as a cultural and educational hub. As Esslingen grows, maintaining infrastructure development, housing availability, and sustainable urban planning will be crucial to sustaining its quality of life and ensuring continued prosperity for its residents.

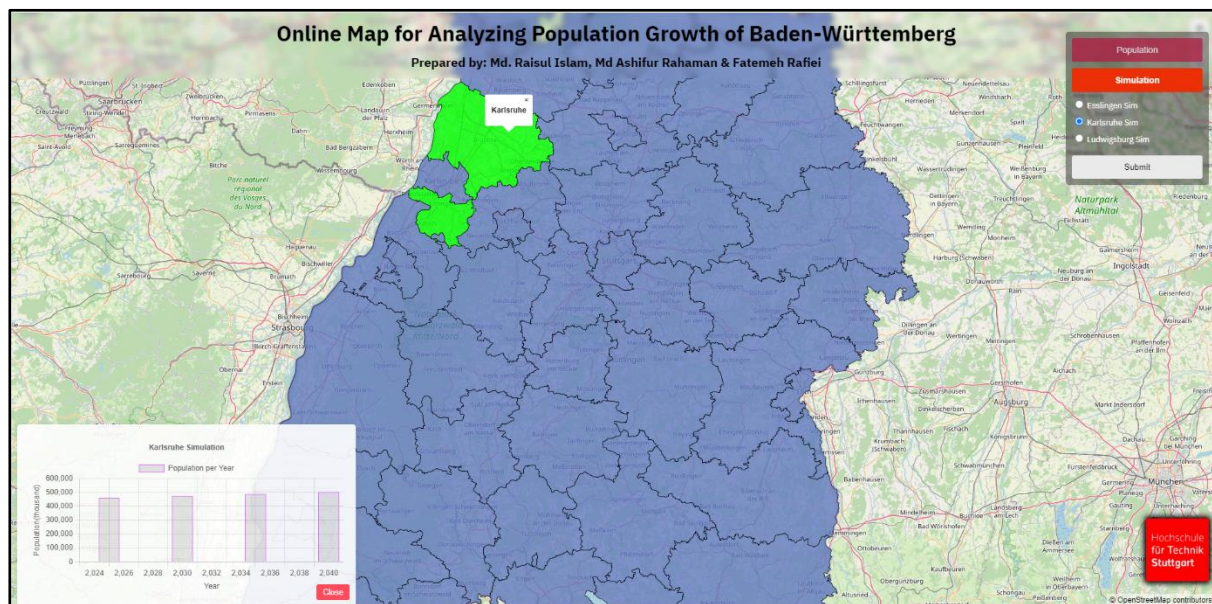


Figure 34: Population Simulation for Karlsruhe (from year 2022 to 2040)

The population projection for Karlsruhe (Figure 34) shows a steady upward trend following an exponential growth formula. Starting from 462,155 in 2025 and reaching an estimated 501,787 by 2040, these figures illustrate sustained population expansion over the next two decades. This growth trajectory reflects Karlsruhe's appeal as a dynamic hub for technology, education, and culture, attracting both domestic migrants and international residents seeking opportunities in its thriving sectors.

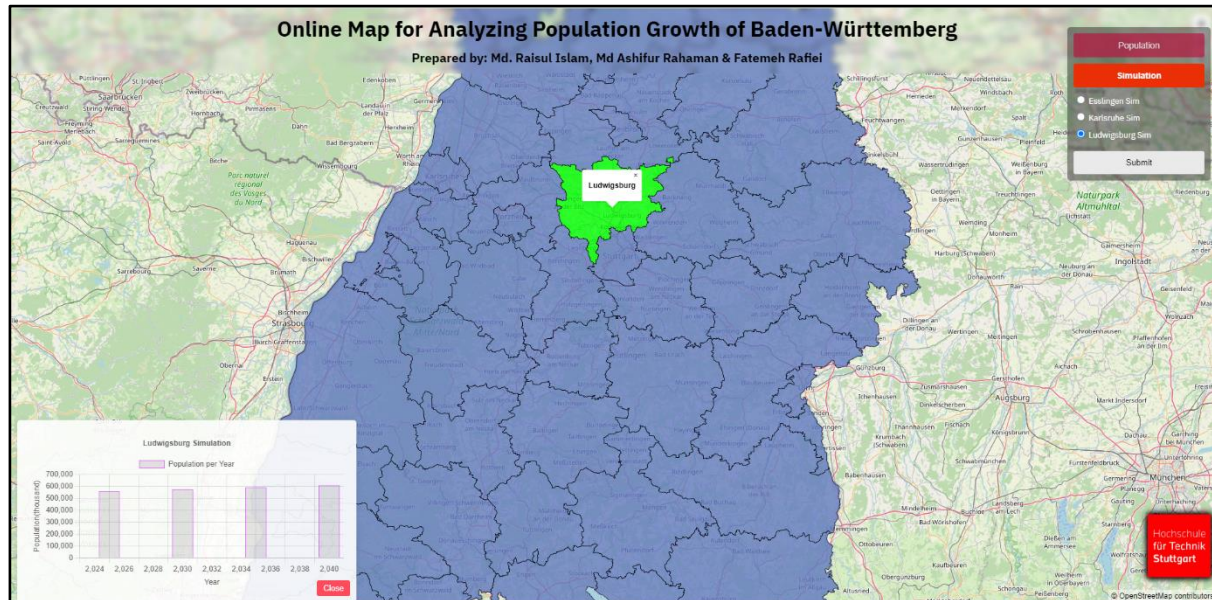


Figure 35: Population Simulation for Ludwigsburg (from year 2022 to 2040)

Figure 35 underscore Ludwigsburg's status as a flourishing urban center within the Stuttgart metropolitan area. The city's attractiveness is bolstered by its historical significance, cultural amenities, educational institutions, and proximity to economic opportunities in Stuttgart. This population expansion indicates continued residential and commercial development, supported by ongoing infrastructure improvements and urban planning initiatives.

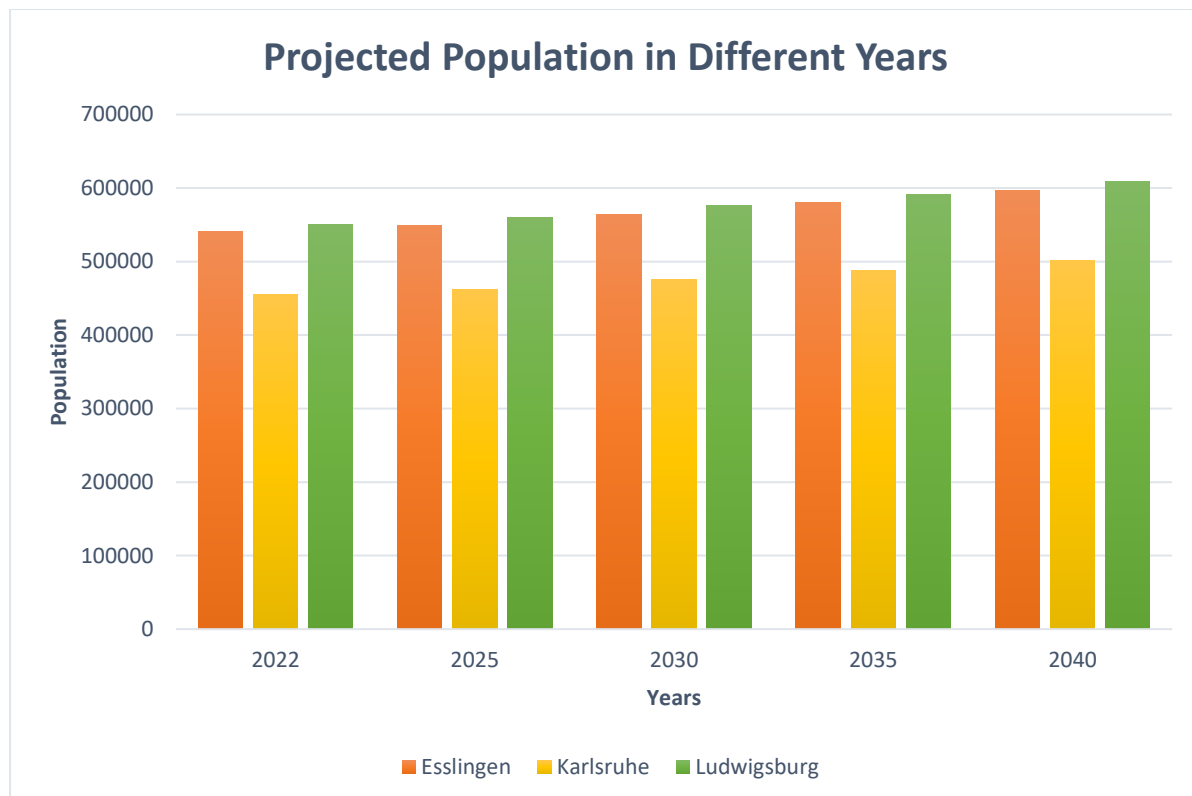


Figure 36: Population Simulation Comparison among the 3 Districts

The population simulation projections for 2025 and 2040 reveal notable trends among Karlsruhe, Esslingen, and Ludwigsburg. By 2025, Karlsruhe's population is projected to be 462,155, Esslingen's population is expected to reach 549,189, and Ludwigsburg's population is estimated at 560,193. Moving forward to 2040, Karlsruhe's population is projected to increase to 501,787, Esslingen's to 596,284, and Ludwigsburg's to 608,232.

A significant observation from these projections is that Ludwigsburg consistently shows the highest population growth among the three cities, while Karlsruhe maintains the lowest population figures throughout the period. This pattern suggests that Ludwigsburg is experiencing a higher rate of attractiveness and development, potentially due to its economic opportunities, cultural amenities, and proximity to Stuttgart, making it a preferred residential choice.

In contrast, despite being a major city with significant technological and educational sectors, Karlsruhe's lower projected population might indicate slower growth rates compared to its counterparts, possibly due to factors such as housing constraints, economic conditions, or competition from neighboring urban centers.

Esslingen's population projections place it between the two, showing steady growth but not surpassing Ludwigsburg. This indicates a balanced development trajectory, benefiting from its industrial base and scenic location, yet not growing as rapidly as Ludwigsburg.

Overall, these simulations highlight the dynamic demographic changes and varying growth rates among the three cities, reflecting their unique economic, cultural, and geographical influences on population trends. Understanding these patterns is crucial for urban planning and resource allocation to address the needs of their growing populations effectively.

6. Challenges, Limitations and Roadmap for Future Development

The completion of the 'Online Map Creation for Population Growth Analysis' project presented several challenges and limitations that shaped our development process. Initially, we opted for Django and Leaflet for backend and frontend development, respectively. However, due to limited familiarity with Django and challenges encountered with Leaflet integration, we transitioned to Flask for backend operations and Maplibre for frontend map rendering. This shift allowed us to leverage a more manageable framework setup conducive to our project goals.

Another significant hurdle involved data preprocessing issues upon downloading statistical data. We encountered formatting inconsistencies such as commas and semicolons, necessitating thorough data cleaning and processing before integration into our system. This added complexity required meticulous attention to ensure data accuracy and consistency throughout the project.

Additionally, implementing interactive features such as radio buttons for map filters and chart displays posed considerable trial and error. Initially, these components did not function as expected, requiring iterative testing and debugging to achieve desired functionality and user experience. Moreover, accessing chart data directly from our PostgreSQL database initially presented challenges. We encountered initial issues with data retrieval and integration, which required troubleshooting and refinement of database queries to ensure seamless data visualization and analysis capabilities.

Throughout these challenges, our team's perseverance and problem-solving skills were essential in overcoming technical hurdles and delivering a robust online mapping tool for population growth analysis. Moving forward, we aim to further optimize functionality and user interface enhancements based on feedback and continued iteration.

Roadmap for Future Development: If someone want to do further development of the project, here are some clues which can help for future development of the projects-

- Enable direct data extraction from the website for seamless access and integration into the mapping tool.
- Implement population growth rate calculations and enable comparative analysis between multiple districts or regions.
- Introduce choropleth maps to enhance visual representation and analysis of population density and growth trends.
- Expand interactive features by adding more options such as selectable year ranges, additional district selections, and diverse datasets.

7. Conclusion

The GIS Studio Project centered on developing an online map to analyze population growth in Stuttgart through the exploration of publicly available data sources. The development of the Online Map Creation for Population Growth Analysis project has successfully achieved its objectives of creating an interactive dashboard interface to visualize and analyze population growth trends in Esslingen, Karlsruhe, and Ludwigsburg. Utilizing Flask for backend development and Maplibre for frontend map rendering, coupled with data sourced from the Landesamt Baden-Württemberg, enabled us to present insightful maps, graphs, and charts that provide a comprehensive view of demographic changes over time.

Throughout this project, we encountered various challenges and limitations, including initial difficulties with software familiarity and data preprocessing issues. These hurdles were addressed through strategic adjustments in our development approach and meticulous data cleaning processes. Looking forward, the roadmap for future development includes enhancing data accessibility directly from the website, integrating population growth rate comparisons, implementing choropleth maps for improved visualization, and expanding interactive features to include selectable year ranges and additional district selections.

This project underscores the significance of open data integration, web mapping technologies, and database management techniques in facilitating informed decision-making and urban planning. By leveraging these tools, stakeholders can gain valuable insights into population dynamics, aiding in policy formulation and resource allocation for sustainable development.

In conclusion, the Online Map Creation for Population Growth Analysis project has successfully achieved its objectives and establishes a solid foundation for advancing spatial data visualization and analysis through geoinformatics and web application efforts. As we refine and expand this tool, it holds promise for making substantial contributions to geoinformatics by enhancing the understanding of population dynamics and supporting informed decision-making processes related to urban development and resource allocation.

8. References

1. Preston, S. H., Heuveline, P., & Guillot, M. (2001). *Demography: Measuring and Modeling Population Processes*. Blackwell Publishers.
2. Goodchild, M. F. (2007). Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4), 211-221.
3. MapLibre (2020). MapLibre: An open-source mapping library. Retrieved from MapLibre
4. Few, S. (2006). *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media.
5. Kitchin, R. (2014). *The Data Revolution: Big Data, Open Data, Data Infrastructures and Their Consequences*. Sage Publications.
6. Reher, D. S. (2011). Economic and social implications of demographic change. *Population and Development Review*, 37(Supplement), 11-33.
7. Janssen, M., Charalabidis, Y., & Zuiderwijk, A. (2012). Benefits, adoption barriers and myths of open data and open government. *Information Systems Management*, 29(4), 258-268.
8. Smith, S. K., Tayman, J., & Swanson, D. A. (2013). *A Practitioner's Guide to State and Local Population Projections*. Springer.
9. Bongaarts, J., & Bulatao, R. A. (Eds.). (2000). *Beyond Six Billion: Forecasting the World's Population*. National Academy Press.
10. Smith, J. (2021). Interactive mapping for urban growth analysis using MapLibre. *Journal of Urban Planning*, 45(3), 456-478.

9. Appendix

There are some screenshots of the codes of the project

From App.py

```
app.py 1 x
app.py > get_espop_data > features
1 from flask import Flask, render_template, jsonify, g
2 import psycopg2
3 import geojson
4
5 app = Flask(__name__)
6
7 connection = psycopg2.connect(
8     user="postgres",
9     password="postgres",
10    host="localhost",
11    port="5432",
12    database="GSS"
13 )
14
15 print('Connection Success')
16 cur = connection.cursor()
17
18 @app.route("/map.html")
19 def get_map():
20     return render_template("index.html")
21
22 @app.route("/stats.html")
23 def get_stats():
24     return render_template("chart.html")
25
26 @app.route('/geojson')
27 def get_geojson_data():
28     cur.execute(
29         """SELECT id, name, schlüssel, ST_AsGeoJSON(geom) AS geometry FROM data;""")
30     multipolygon_geojson = cur.fetchall()
31
32     features = []
33     index = 1
34     for row in multipolygon_geojson:
35         geometry = geojson.loads(row[3])
36         feature = {
37             "type": "Feature",
38             "geometry": geometry,
39             "id": index,
40             "properties": {
41                 'id': row[0],
42                 'name': row[1],
43                 'schlüssel': row[2]
44             }
45         }
46         features.append(feature)
47
48     index +=1
```

```
app.py > get_espod_data > features
27 def get_geojson_data():
49
50     feature_collection = {
51         "type": "FeatureCollection",
52         "features": features
53     }
54
55     return jsonify(feature_collection)
56
57 #Chat Creation
58 @app.route('/population/espod')
59 def get_espod_data():
60     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
61     cur.execute("select distinct on (year) 'Esslingen Population' as title, year, poppersqkm from espod order by year;")
62     data = cur.fetchall()
63
64     features = []
65
66     for row in data:
67         feature = {
68             "title": row[0],
69             "year": row[1],
70             "population": row[2],
71         }
72
73         features.append(feature)
74
75     return jsonify(features)
76
77 @app.route('/population/karlpod')
78 def get_karlpod_data():
79     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
80     cur.execute("select distinct on (year) 'Karlsruhe Population' as title, year, poppersqkm from karlpod order by year;")
81     data = cur.fetchall()
82
83     features = []
84
85     for row in data:
86         feature = {
87             "title": row[0],
88             "year": row[1],
89             "population": row[2],
90         }
91
92         features.append(feature)
93
94     return jsonify(features)
95
```

```

app.py 1 x
app.py > get_espod_data > features
96 @app.route('/population/ludpop')
97 def get_ludpop_data():
98     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
99     cur.execute('select distinct on (year) 'Ludwigsburg Population' as title, year, poppersqkm from ludpop order by year;')
100     data = cur.fetchall()
101
102     features = []
103
104     for row in data:
105         feature = {
106             "title": row[0],
107             "year": row[1],
108             "population": row[2],
109         }
110
111         features.append(feature)
112
113     return jsonify(features)
114
115
116 # Simulated data
117 #Chat Creation
118 @app.route('/simulation/espod')
119 def get_essim_data():
120     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
121     cur.execute("SELECT cityname || ' Simulated' as title, year::int, pop::int FROM popprojection WHERE cityname = 'Esslingen' ORDER BY year;")
122     data = cur.fetchall()
123
124     features = []
125
126     for row in data:
127         feature = {
128             "title": row[0],
129             "year": row[1],
130             "population": row[2],
131         }
132
133         features.append(feature)
134
135     return jsonify(features)
136
137
138 @app.route('/simulation/karlpop')
139 def get_karlsim_data():
140     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
141     cur.execute("SELECT cityname || ' Simulated' as title, year::int, pop::int FROM popprojection WHERE cityname = 'Karlsruhe' ORDER BY year;")
142     data = cur.fetchall()
143
144     features = []
145
146     for row in data:
147         feature = {
148             "title": row[0],
149             "year": row[1],
150             "population": row[2],
151         }
152
153         features.append(feature)
154
155     return jsonify(features)
156
157 @app.route('/simulation/ludpop')
158 def get_ludsim_data():
159     #cur.execute('SELECT category, COUNT(*) FROM your_table GROUP BY category')
160     cur.execute("SELECT cityname || ' Simulated' as title, year::int, pop::int FROM popprojection WHERE cityname = 'Ludwigsburg' ORDER BY year;")
161     data = cur.fetchall()
162
163     features = []
164
165     for row in data:
166         feature = {
167             "title": row[0],
168             "year": row[1],
169             "population": row[2],
170         }
171
172         features.append(feature)
173
174     return jsonify(features)
175
176 if __name__ == '__main__':
177     # app.run(debug=True, port=5000)
178     app.run(debug=True)

```

Index.html

```

index.html x
templates > index.html > html > body > a
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Population Data Portal</title>
6 <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no" />
7
8 <!-- Maplibre GL -->
9 <link href="https://unpkg.com/maplibre-gl@2.4.0/dist/maplibre-gl.css" rel="stylesheet" />
10
11 <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
12 <link rel="stylesheet"
13 href="https://fonts.googleapis.com/css2?family=IBM+Plex+Sans:wght@200;300;400;500;600;700&display=swap">
14 <link rel="stylesheet" href="{{ url_for('static', filename='css/index.css') }}">
15 <meta name="viewport" content="width=device-width, initial-scale=1.0">
16 <title>University Logo</title>
17 </head>
18 <body>
19 <div class="titleDiv">
20 <h1>Online Map for Analyzing Population Growth of Baden-Württemberg</h1>
21 <h2>Prepared by: Md. Raisul Islam, Md Ashifur Rahaman & Fatemeh Rafiei</h2>
22 </div>
23 <ul>
24 {% for row in data %}
25 <li>{{ row }}</li>
26 {% endfor %}
27 </ul>
28 <div id="map"></div>
29 <div class="credit"></div>
30 <div class="control-container">
31 <button id="populationBtn" class="active">Population</button>
32 <button id="simulationBtn">Simulation</button>
33 <div id="populationOptions" class="options">
34 <label><input type="radio" name="population" id="espop" value="espop"> Esslingen Pop</label>
35 <label><input type="radio" name="population" id="karlpop" value="karlpop"> Karlsruhe Pop</label>
36 <label><input type="radio" name="population" id="ludpop" value="ludpop"> Ludwigsburg Pop</label>
37 </div>
38 <div id="simulationOptions" class="options hidden">
39 <label><input type="radio" name="simulation" id="essim" value="espop"> Esslingen Sim</label>
40 <label><input type="radio" name="simulation" id="karlsim" value="karlpop"> Karlsruhe Sim</label>
41 <label><input type="radio" name="simulation" id="ludsim" value="ludpop"> Ludwigsburg Sim</label>
42 </div>
43 <button id="submitBtn">Submit</button>
44 </div>
45 <div id="chartContainer" class="hidden">
46 <button id="closeChartBtn">Close</button>
47 <canvas id="myChart"></canvas>
48 </div>
49 <script src="https://code.jquery.com/jquery-3.7.1.min.js"
50 integrity="sha256-/JqT3SQfawRcv/BIHPTkBs00EvFFmqPF/1YI/Cxo=" crossorigin="anonymous"></script>
51 <script src="https://unpkg.com/maplibre-gl@2.4.0/dist/maplibre-gl.js"></script>
52 <script src="{{ url_for('static', filename='js/main.js') }}"></script>
53 <a href="https://www.hft-stuttgart.de/" target="_blank">
54 <div class="credit"></div>
55 </a>
56 </body>
57 </html>

```

Main.js

```

1  static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
2  1
3  2 var maplibreMap = new maplibregl.Map({
4  3   container: 'map', // container ID
5  4   style: {
6  5     'version': 8,
7  6     'sources': {
8  7       'osm-tiles': {
9  8         'type': 'raster',
10 9         'tiles': [
1110          'https://a.tile.openstreetmap.org/{z}/{x}/{y}.png',
1211          'https://b.tile.openstreetmap.org/{z}/{x}/{y}.png',
1312          'https://c.tile.openstreetmap.org/{z}/{x}/{y}.png'
1413        ],
1514        'tileSize': 256,
1615        'attribution': '© OpenStreetMap contributors'
1716      }
1817    },
1918    'layers': [
2019      {
2120        'id': 'osm-tiles',
2221        'type': 'raster',
2322        'source': 'osm-tiles',
2423        'minzoom': 0,
2524        'maxzoom': 19
2625      }
2726    ]
2827  },
2928  center: [9.1544, 48.6361], // starting position [lng, lat]
3029  zoom: 8 // starting zoom
3130 });
3231 // Add zoom and rotation controls to the map.
3332 maplibreMap.addControl(new maplibregl.NavigationControl());
3433
3534
3635 var geojsonUrl = "http://localhost:5000/geojson";
3736
3837 maplibreMap.on('load', async function () {
3938   maplibreMap.addSource('geojson_data', {
4039     'type': 'geojson',
4140     'data': geojsonUrl
4241   });
4342
4443
4544   // Add a click event listener to the layer
4645   maplibreMap.addLayer({
4746     'id': 'geojson_layer',
4847     'type': 'fill',
4948     'source': 'geojson_data',
5049     'layout': {},
5150     'paint': {
5251       // 'fill-color': ['case', ['boolean', ['feature-state', 'clicked'], false], '#FFF000', '#627BC1'],
5352       'fill-color': [
5453         'case',
5554         ['boolean', ['feature-state', 'clicked'], false], '#00FF00', // clicked color
5655         ['boolean', ['feature-state', 'hover'], false], '#FFF000', // hover color
5756         '#627BC1' // default color
5857       ],
5958     }
5959   },

```

```

JS main.js x
static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
38 maplibreMap.on('load', async function () {
63
64     maplibreMap.addLayer({
65         'id': 'geojson_layer_outline',
66         'type': 'line',
67         'source': 'geojson_data',
68         'layout': {},
69         'paint': {
70             'line-color': '#000000',
71             'line-width': 0.5
72         }
73     });
74
75
76     // Variable to hold the ID of the currently hovered feature
77     var hoveredFeatureId = null;
78
79     // Event listener for click to log the properties of the clicked feature
80     maplibreMap.on('click', 'geojson_layer', function (e) {
81         if (e.features && e.features.length > 0) {
82             var clickedFeature = e.features[0];
83             var name = clickedFeature.properties.name
84             console.log(name);
85             if (name == "Esslingen" || name == "Ludwigsburg" || name == "Karlsruhe") {
86                 console.log('Population data available for ${name}')
87             }
88         }
89     });
90
91     // Event listener for mouse move to update the feature state for hover
92     maplibreMap.on('mousemove', 'geojson_layer', function (e) {
93         if (e.features.length > 0) {
94             if (hoveredFeatureId) {
95                 maplibreMap.setFeatureState(
96                     { source: 'geojson_data', id: hoveredFeatureId },
97                     { hover: false }
98                 );
99             }
100
101             hoveredFeatureId = e.features[0].id;
102
103             maplibreMap.setFeatureState(
104                 { source: 'geojson_data', id: hoveredFeatureId },
105                 { hover: true }
106             );
107         }
108     });
109
110     // Event listener for mouse leave to reset the hover state
111     maplibreMap.on('mouseleave', 'geojson_layer', function () {
112         if (hoveredFeatureId) {
113             maplibreMap.setFeatureState(
114                 { source: 'geojson_data', id: hoveredFeatureId },
115                 { hover: false }
116             );
117         }
118         hoveredFeatureId = null;
119     });
120

```

```

w Go Run Terminal Help
JS main.js x
static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
38 maplibreMap.on('load', async function () {
124 // Add a click event listener to the layer
125 maplibreMap.on('click', 'geojson_layer', function (e) {
126     if (clickedFeatureId) {
127         maplibreMap.setFeatureState(
128             { source: 'geojson_data', id: clickedFeatureId },
129             { clicked: false }
130         );
131     }
132
133     clickedFeatureId = e.features[0].id;
134
135     maplibreMap.setFeatureState(
136         { source: 'geojson_data', id: clickedFeatureId },
137         { clicked: true }
138     );
139
140     var coordinates = e.lngLat;
141     var properties = e.features[0].properties;
142
143     // Log the properties to the console
144     console.log(properties);
145
146     // Create a popup
147     var popupContent = '<b>' + properties.name + '</b>';
148     new maplibregl.Popup()
149         .setLngLat(coordinates)
150         .setHTML(popupContent)
151         .addTo(maplibreMap);
152     });
153
154     // Reset the color when clicking outside of the features
155     maplibreMap.on('click', function (e) {
156         var features = maplibreMap.queryRenderedFeatures(e.point, {
157             layers: ['geojson_layer']
158         });
159
160         if (!features.length && clickedFeatureId) {
161             maplibreMap.setFeatureState(
162                 { source: 'geojson_data', id: clickedFeatureId },
163                 { clicked: false }
164             );
165             clickedFeatureId = null;
166         }
167     });
168
169 });
170
171 // static/js/script.js
172 document.addEventListener('DOMContentLoaded', function () {
173     const populationBtn = document.getElementById('populationBtn');
174     const simulationBtn = document.getElementById('simulationBtn');
175     const populationOptions = document.getElementById('populationOptions');
176     const simulationOptions = document.getElementById('simulationOptions');
177     const submitBtn = document.getElementById('submitBtn');
178     const chartContainer = document.getElementById('chartContainer');
179     const closeChartBtn = document.getElementById('closeChartBtn');
180     const ctx = document.getElementById('myChart').getContext('2d');
181     let mvChart:

```



```

JS main.js x
static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
172 document.addEventListener('DOMContentLoaded', function () {
183   populationBtn.addEventListener('click', function () {
186     populationOptions.classList.remove('hidden');
187     simulationOptions.classList.add('hidden');
188   });
189
190   simulationBtn.addEventListener('click', function () {
191     simulationBtn.classList.add('active');
192     populationBtn.classList.remove('active');
193     simulationOptions.classList.remove('hidden');
194     populationOptions.classList.add('hidden');
195   });
196
197   submitBtn.addEventListener('click', function () {
198     let apiUrl = '';
199     let title = '';
200     if (populationBtn.classList.contains('active')) {
201       if (document.getElementById('espop').checked) {
202         apiUrl = 'http://localhost:5000/population/espop';
203         title = 'Esslingen Population';
204       } else if (document.getElementById('karlpop').checked) {
205         apiUrl = 'http://localhost:5000/population/karlpop';
206         title = 'Karlsruhe Population';
207       } else if (document.getElementById('ludpop').checked) {
208         apiUrl = 'http://localhost:5000/population/ludpop';
209         title = 'Ludwigsburg Population';
210       }
211     } else if (simulationBtn.classList.contains('active')) {
212       if (document.getElementById('essim').checked) {
213         apiUrl = 'http://localhost:5000/simulation/espop';
214         title = 'Esslingen Simulation';
215       } else if (document.getElementById('karlsim').checked) {
216         apiUrl = 'http://localhost:5000/simulation/karlpop';
217         title = 'Karlsruhe Simulation';
218       } else if (document.getElementById('ludsim').checked) {
219         apiUrl = 'http://localhost:5000/simulation/ludpop';
220         title = 'Ludwigsburg Simulation';
221       }
222     }
223
224     if (apiUrl) {
225       fetch(apiUrl)
226         .then(response => response.json())
227         .then(data => {
228           console.log('Success:', data);
229           // Prepare data for the chart
230           const labels = data.map(item => (item.year));
231           const values = data.map(item => item.population);
232
233           // Destroy previous chart instance if it exists
234           if (myChart) {
235             myChart.destroy();
236           }
237
238           // Create a new chart
239           myChart = new Chart(ctx, {
240             type: 'bar',
241             data: {
242               labels: labels,

```

```

tion View Go Run Terminal Help
... JS main.js x
static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
172 document.addEventListener('DOMContentLoaded', function () {
197 submitBtn.addEventListener('click', function () {
227 .then(data => {
232
233 // Destroy previous chart instance if it exists
234 if (myChart) {
235 myChart.destroy();
236 }
237
238 // Create a new chart
239 myChart = new Chart(ctx, {
240 type: 'bar',
241 data: {
242 labels: labels,
243 datasets: [{
244 label: 'Population per Year',
245 data: values,
246 borderColor: '#e056fd',
247 borderWidth: 1,
248 fill: false
249 }]
250 },
251 options: {
252 responsive: true,
253 scales: {
254 x: {
255 type: 'linear',
256 position: 'bottom',
257 title: {
258 display: true,
259 text: 'Year'
260 }
261 },
262 y: {
263 beginAtZero: true, // Begin y-axis at 0
264 title: {
265 display: true,
266 text: 'Population(thousand)'
267 }
268 },
269 },
270 plugins: {
271 title: {
272 display: true,
273 text: title
274 }
275 },
276 },
277 });
278
279 })
280 .catch(error => {
281 console.error('Error:', error);
282 });
283 } else {
284 alert('Please select an option.');
```

```

JS main.js x
static > js > JS main.js > maplibreMap.on('load') callback > maplibreMap.on('click', 'geojson_layer') callback
172 document.addEventListener('DOMContentLoaded', function () {
197 submitBtn.addEventListener('click', function () {
206 });
286
287
288 $(document).ready(function () {
289 // Hide chart container on page load
290 $('#chartContainer').hide();
291
292 // Show chart container on submit button click
293 $('#submitBtn').click(function () {
294 // Your existing submit button logic goes here
295
296 // Show the chart container
297 $('#chartContainer').show();
298 });
299
300 // Close chart container on close button click
301 $('#closeChartBtn').click(function () {
302 // Hide the chart container
303 $('#chartContainer').hide();
304 });
305 });
306
307 document.querySelectorAll('input[name="population"]').forEach((elem) => {
308 elem.addEventListener('change', (event) => {
309 let selectedValue = event.target.nextSibling.textContent.trim().replace(/ (Pop|Sim)/, '');
310 highlightFeature(selectedValue);
311 });
312 });
313
314 document.querySelectorAll('input[name="simulation"]').forEach((elem) => {
315 elem.addEventListener('change', (event) => {
316 let selectedValue = event.target.nextSibling.textContent.trim().replace(/ (Pop|Sim)/, '');
317 highlightFeature(selectedValue);
318 });
319 });
320
321 document.getElementById('submitBtn').addEventListener('click', () => {
322 unhighlightFeature();
323 });
324
325 function highlightFeature(name) {
326 maplibreMap.setFilter('geojson_layer', ['==', ['get', 'name'], name]);
327 }
328
329 function unhighlightFeature() {
330 maplibreMap.setFilter('geojson_layer', null);
331 }
332
333 });
334

```

Index.css

```

# index.css x
static > css > # index.css > h1
1  /* static/css/styles.css */
2  v body {
3      margin: 0;
4      padding: 0;
5      overflow: hidden;
6      font-family: 'IBM Plex Sans', sans-serif;
7  }
8
9  v #map {
10     position: absolute;
11     width: 100%;
12     height: 100vh;
13     background-color: #eaeaea; /* Placeholder for the map */
14     z-index: 1;
15     top: 0;
16 }
17
18 v .control-container {
19     position: absolute;
20     color: #fff;
21     top: 30px;
22     right: 10px;
23     background: rgba(0, 0, 0, 0.4);
24     padding: 10px;
25     border-radius: 5px;
26     box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
27     z-index: 1000;
28     backdrop-filter: blur(2px);
29     width: 13rem;
30 }
31
32 v #populationBtn, #simulationBtn {
33     display: block;
34     width: 100%;
35     margin-bottom: 10px;
36     padding: 10px;
37     background-color: #b71540;
38     color: #fff;
39     border: 1px solid transparent;
40     border-radius: 3px;
41     cursor: pointer;
42     opacity: 0.7;
43     font-size: 14px;
44 }
45
46 v #populationBtn.active, #simulationBtn.active {
47     background-color: #eb2f06;
48     font-weight: 600;
49     opacity: 1;
50 }
51
52 v #submitBtn {
53     display: block;
54     width: 100%;
55     margin: 10px 0 0 0;
56     padding: 10px;
57     background-color: #eaeaea;
58     color: #000;
59     border: 1px solid transparent;

```

```

ew Go Run Terminal Help
# index.css x
static > css > # index.css > h1
52 #submitBtn {
53     display: block;
54     width: 100%;
55     margin: 10px 0 0 0;
56     padding: 10px;
57     background-color: #eaeaea;
58     color: #000;
59     border: 1px solid transparent;
60     border-radius: 3px;
61 }
62
63 #submitBtn:hover {
64     background-color: #78e08f;
65     cursor: pointer;
66     transition: 0.3s linear;
67 }
68
69 .options {
70     display: flex;
71     flex-direction: column;
72 }
73
74 .options label {
75     margin-bottom: 7px;
76     font-size: 13px;
77 }
78
79 .hidden {
80     display: none;
81 }
82
83 #chartContainer {
84     position: absolute;
85     width: 500px;
86     height: 250px;
87     left: 10px;
88     bottom: 0;
89     background: rgba(255, 255, 255, 0.9);
90     padding: 20px;
91     border-radius: 5px;
92     box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
93     z-index: 1000; /* Ensure chart is on top */
94     display: flex;
95     flex-direction: column;
96     align-items: center;
97 }
98
99 #closeChartBtn {
100     background-color: #ff4b5c;
101     color: #fff;
102     border: none;
103     border-radius: 3px;
104     cursor: pointer;
105     padding: 5px 10px;
106     position: absolute;
107     bottom: 10px;
108     right: 10px;
109     font-size: 0.8em;
110 }

```

```
# index.css X
static > css > # index.css > h1
110 }
111 .credit {
112     width: 100px;
113     height: 100px;
114     overflow: hidden;
115     position: fixed;
116     z-index: 99999;
117     background: url(../Images/hft.jpg) no-repeat;
118     background-size: 100px;
119     bottom: 40px;
120     right: 10px;
121     box-shadow: 0 0 10px 1px #000;
122 }
123
124 .titleDiv {
125     position: absolute;
126     top: 0;
127     align-items: center;
128     justify-content: center;
129     padding: 5px;
130     backdrop-filter: blur(5px);
131     z-index: 10;
132     display: flex;
133     flex-direction: column;
134     width: 100%;
135 }
136
137 h1 {
138     text-align: left;
139     margin: 10px;
140     padding: 0;
141     font-size: 34px;
142     line-height: 1;
143     color: #000;
144     z-index: 2;
145     position: relative;
146 }
147
148 h2 {
149     text-align: left;
150     margin: 10px;
151     padding: 0;
152     font-size: 20px;
153     line-height: 1;
154     background-color: none;
155     color: #000;
156     z-index: 2;
157     position: relative;
158 }
159
160
```