THE UNIVERSITY OF DANANG

**UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**FACULTY OF ADVANCED SCIENCE AND TECHNOLOGY**

|  |  |
| --- | --- |
|  |  |

**CAPSTONE PROJECT REPORT**

**M2 REPORT**

**DESIGN AND DEVELOPMENT OF A DATA VISUALIZATION LIBRARY FOR SPARQL QUERIES**

|  |  |  |
| --- | --- | --- |
| **Student:** |  | **Minh Huy Do Class: 20PFIEV3** |

|  |  |  |
| --- | --- | --- |
| **Academic advisor:**  **Supervisor:**  **Co-Supervisor:** |  | **Dr. Thi My Hanh Le**  **Assoc. Prof., Dr. Aline Menin**  **Prof., Dr. Catherine Faron** |

**Danang, 05 / 2025**

**Contents**

[Chapter 1 Overview 4](#_Toc197426756)

[Chapter 2 Proposed methods 5](#_Toc197426757)

[2.1 Proposed techniques and solution: 5](#_Toc197426758)

[2.2 Required tools and materials 6](#_Toc197426759)

[2.2.1 Programming language: 6](#_Toc197426760)

[2.2.2 Libraries and frameworks: 7](#_Toc197426761)

[2.2.3 Data resources: (SPARQL Endpoints): 7](#_Toc197426762)

[2.2.4 Development Environment: 7](#_Toc197426763)

[2.3 Accomplished tasks 8](#_Toc197426764)

[Chapter 3 Up-to-date obtained results 11](#_Toc197426808)

[3.1 Bar Chart 11](#_Toc197426812)

[3.1.1 Regular Bar Chart 11](#_Toc197426813)

[3.1.2 Stacked Bar Chart 13](#_Toc197426823)

[3.1.3 Percentage Stacked Bar Chart 15](#_Toc197426831)

[3.1.4 Grouped Bar Chart 16](#_Toc197426839)

[3.2 Pie Chart 18](#_Toc197426846)

[3.3 Geographic Map 19](#_Toc197426853)

[3.3.1 Choropleth Map 19](#_Toc197426854)

[3.3.2 Connection Map 21](#_Toc197426862)

[3.3.3 Bubble Map 22](#_Toc197426868)

[3.3.4 Hexbin Map 23](#_Toc197426873)

[3.4 Color Palettes 24](#_Toc197426878)

[3.4.1 Purpose 24](#_Toc197426879)

[3.4.2 Operating principle 25](#_Toc197426883)

[3.4.3 Syntax 25](#_Toc197426895)

[3.4.3.1 Categorical schemes 25](#_Toc197426896)

[3.4.3.2 Cyclical schemes 26](#_Toc197426908)

[3.4.3.3 Diverging schemes 26](#_Toc197426911)

[3.4.3.4 Sequential schemes 28](#_Toc197426930)

[Chapter 4 Plan for Implementation 32](#_Toc197426976)

[**References 33**](#_Toc197426990)

**Tables**

[Table 2.1 Accomplished tasked in M1 & M2 10](#_Toc197428094)

[Table 4.1 Tasks to do in M3 32](#_Toc197428095)

**Figures**

[Figure 3.1 Metadata of Regular Bar Chart 12](#_Toc197428161)

[Figure 3.2 Visualization of Regular Bar Chart 13](#_Toc197428162)

[Figure 3.3 Metadata of Stacked Bar Chart 14](#_Toc197428163)

[Figure 3.4 Visualization of Stacked Bar Chart 15](#_Toc197428164)

[Figure 3.5 Metadata of Percentage Stacked Bar Chart 16](#_Toc197428165)

[Figure 3.6 Visualization of Percentage Stacked Bar Chart 16](#_Toc197428166)

[Figure 3.7 Metadata of Grouped Bar Chart 17](#_Toc197428167)

[Figure 3.8 Visualization of Grouped Bar Chart 18](#_Toc197428168)

[Figure 3.9 Metadata of Pie Chart 18](#_Toc197428169)

[Figure 3.10 Visualization of Pie Chart 19](#_Toc197428170)

[Figure 3.11 Metadata of Choropleth Map 20](#_Toc197428171)

[Figure 3.12 Visualization of Choropleth Map 21](#_Toc197428172)

[Figure 3.13 Metadata of Connection Map 22](#_Toc197428173)

[Figure 3.14 Visualization of Connection Map 22](#_Toc197428174)

[Figure 3.15 Visualization of Bubble Map 23](#_Toc197428175)

[Figure 3.16 Visualization of Hexbin Map 24](#_Toc197428176)

# Overview

In today's data-driven world, vast and complex datasets are increasingly published using RDF (Resource Description Framework) [1] and made available as LOD (Linked Open Data) [2] These datasets hold immense potential for decision-making across various fields, but their value lies in the ability to effectively analyze and visualize the information they contain. Custom visualizations, while powerful, are often hard-coded, requiring specialized programming skills and making them challenging to reuse across different applications. Additionally, these solu ons lack transparency, making it difficult to understand how visualization and interactivity are configured. Simplifying the creation of visualizations for non-programmers, while taking into account both data structures and user intent, is therefore essential. Despite advancements, most exis ng visualiza on approaches are tailored for datasets in formats like CSV or JSON, overlooking the unique complexies of RDF datasets.

The objective of the project is to design and develop a library to simplify and enhance the creation of visualizations for SPARQL [3] queries, making them accessible to semantic web experts and decision makers without the need for advanced programming skills. Develop a visualization library inspired by tools such as ECharts [4], MGExplorer [5], and D3.js [6]. This library will transform SPARQL queries and visual mapping configurations into interactive, reusable visualizations that can be directly applied to a variety of projects.

# Proposed methods

## 2.1 Proposed techniques and solution:

The proposed solution is a comprehensive, JavaScript-based library designed to empower users with the ability to generate interactive, dynamic, and reusable visualizations from SPARQL query results, all without requiring in-depth programming knowledge. This solution aims to bridge the gap between technical and non-technical users, offering a powerful yet accessible tool for visualizing RDF data. Key features of the library include:

* **Support for JSON-based SPARQL result and diverse chart type:** Instead of directly processing various RDF serialization formats like Turtle or RDF/XML, the library focuses on **SPARQL query results in standardized JSON format (SPARQL Results JSON Format)**. This approach simplifies parsing and enables consistent handling of tabular or triple-based results across different SPARQL endpoints. The use of JSON also allows for easier transformation and mapping into visual elements. Users can then choose from a variety of visualization types—such as **bar charts, pie charts, and network graphs**—based on the nature of the data and the insights they wish to extract. This flexibility ensures that different types of RDF-derived information can be meaningfully and clearly represented.
* **Vega-Lite Grammar - Simplified and flexible customization:** One of the core strengths of the library is its flexible customization, which allows users to easily tailor visualizations to their needs without writing complex code. To achieve this, we decided to use **Vega-Lite grammar**—a high-level declarative language for creating interactive and expressive visualizations. Vega-Lite provides a concise JSON syntax to specify data encodings, transformations, and chart configurations. It enables users to define visualizations like bar charts, scatter plots, and layered views with minimal specification, while still allowing for interactivity and extensibility. Since Vega-Lite compiles to full Vega specifications, users can benefit from both simplicity and underlying power when needed, without having to write pure JavaScript.
* **Web Component – High performance and Scalability Solution:** We decided to use Web Components to build the library because it is a modern, standard and more scalable solution than traditional JavaScript frameworks. The main reasons include:
* **Support for native web standards:** Web Components are natively supported by browsers without the need for external libraries or frameworks, ensuring long-term compatibility.
* **Strong packaging capabilities and high reusability:** Shadow DOM technology isolates the CSS and JavaScript of each component, avoiding conflicts with other parts of the application. The built components can be reused in many different projects without major modifications.
* **High performance:** Since Web Components run directly in the browser without the need for additional runtime, performance is optimized, which is especially important when dealing with large data visualization.
* **Easy integration into any web application:** Web Components can be used directly in HTML without depending on any specific framework, allowing for more flexible integration.

When choosing between MDN Web Component [8] and Stencil [9], we decided to use MDN Web Component for the following reasons: it is based on official web standards, MDN Web Component is highly stable and is not affected by changes in third-party frameworks. Unlike Stencil, MDN Web Components does not require the use of TypeScript or JSX, providing more flexibility in the development process.

## 2.2 Required tools and materials

To develop the SPARQL data visualization library, the necessary tools and materials used are:

### 2.2.1 Programming language:

* JavaScript: JavaScript is the primary programming language used for the development of the library. Its ability to run natively in modern web browsers makes it the ideal choice for a client-side visualization tool. Moreover, JavaScript has a vast ecosystem of libraries and tools that support data manipulation, UI development, and integration with APIs such as SPARQL endpoints.

### 2.2.2 Libraries and frameworks:

* D3.js:
* Powerful JavaScript library for DOM and SVG based data visualization.
* Supports drawing graphs, charts, and tree structures suitable for RDF data.
* Provides interactivity such as zooming, dragging, highlighting data.
* Echarts:
* Dynamic charting library supporting chart types.
* Easily integrate with data from SPARQL to visualize statistical information.
* Web Components:
* Provides the ability to encapsulate UI components for easy reuse.
* Helps build intuitive configuration interfaces.

### 2.2.3 Data resources: (SPARQL Endpoints):

The library is designed to work seamlessly with multiple RDF data sources accessed via SPARQL queries. The key supported endpoints include:

* Dbpedia (<https://dbpedia.org/sparql>): Offers structured data extracted from Wikipedia, suitable for visualizing knowledge graphs related to people, places, organizations, and other real-world entities.
* Wikidata (<https://query.wikidata.org/>): A collaboratively edited knowledge base containing rich, multilingual, semantic data across diverse domains. It provides extensive support for SPARQL queries with customizable filters and joins.
* Custom sources: Users can import SPARQL query results directly in JSON format from local files or private SPARQL endpoints. This enables use of the visualization library in closed environments or with proprietary data systems.

### 2.2.4 Development Environment:

* Visual Studio Code: Visual Studio Code is the primary integrated development environment (IDE) used in the project. It provides: Popular source code editor, supporting many utilities to help write JavaScript code effectively. Currently we are using VS Code but in the future will aim to expand to more frameworks.
* Node & NPM (Node Package Manager): Provides a runtime environment for JavaScript, allowing code to be executed outside the browser. And NPM [10] is a library management system, helping to install and manage dependencies for the project.

## 2.3 Accomplished tasks

|  |  |  |
| --- | --- | --- |
|  | **Tasks** | **Timeline** |
| 1  (M1) | * Do the web semantic’s course. * Study tutorial web components. * Check how to export components at npm package. | 1 week  (06/02 – 13/02) |
| 2  (M1) | * Create some chart’s examples: Pie chart, Bar chart, Node Link Diagram, Choropleth map. * Do the english version course of semantic web. | 4 days  (13/02 – 17/02) |
| 3  (M1) | * Separate each chart components. * Hide data from outside. * Create color picker for charts. * Import the package for new project. * Do the RDF Lab. | 1 week  (18/02 – 25/02) |
| 4  (M1) | * Do the RDF Schema Lab. * Do the SPARQL Lab. | 1 week  (25/02 - 03/03) |
| 5  (M1) | * Apply Pie Chart and Bar Chart with Vega Grammar. * Use SPARQL query’s results. | 1 week  (04/03 - 11/03) |
| 6  (M1) | * Extract the vega specifications for each different charts. * Create legend for charts, add the title for axes of barChart | 1 week  (11/03 – 18/03) |
| 7  (M1) | * Use Vega Lite instead of Vega. * Put the default color for charts if the parameter color is not present. * Apply more subtypes per chart type. * Find the data for Map and Node Link Diagram | 1 weeks  (18/03 – 25/03) |
| 8  (M2) | * Understand how to use Stacked Bar Chart without using « aggregate » in Vega-Lite grammar. * Use Vega-Lite parameters for metadata. * Add a new « direction » parameter to distinguish between vertical and horizontal bar charts. | 1 weeks  (25/03 – 01/04) |
| 9  (M2) | * Analyze how stacked operations work in Vega-Lite, propose and explain new solutions. * Create a Grouped Bar Chart using « xOffset ». * Allow users to define custom color scales using an array of colors. | 9 days  (01/04 – 10/04) |
| 10  (M2) | * Use stack parameter to define bar chart subtype (true = Stacked Bar Chart, false = Grouped Bar Chart, « normalize » = Normalized Stacked Bar Chart). * Change encoding structure from array to object. * Add console warnings for color range mismatches. * Support domain and range settings for color scales. * Allow color palette names or D3 functions. * Research axis scale types: Linear, Pow, Log. | 12 days  (10/04 – 22/04) |
| 11  (M2) | * Improve Grouped Bar Chart display by removing zero-value subgroups and centering bars. * Fix domain/range logic for color:   + Ignore extra domain values not in data.   + Fill missing domains with remaining colors.   + Warn in console with explanation. * Create my own library functions that allow users to input a color name string and connect to D3 to display the color palette (e.g., Paired, Reds[5]). * Display axis scale type as an annotation and raise the max value marker by one level. | 1 week  (22/04 – 29/04) |
| 12  (M2) | * Add axis label rotation via axis.labelAngle in encoding. * Refactor code to remove duplication and reduce size. * Ensure colorDomain appears before colorRange, and both are the same length. * Publish the Bar Chart version to NPM and start testing. | 2 weeks  (29/04 – 13/05) |

*Table 2.1 Accomplished tasked in M1 & M2*

# Up-to-date obtained results

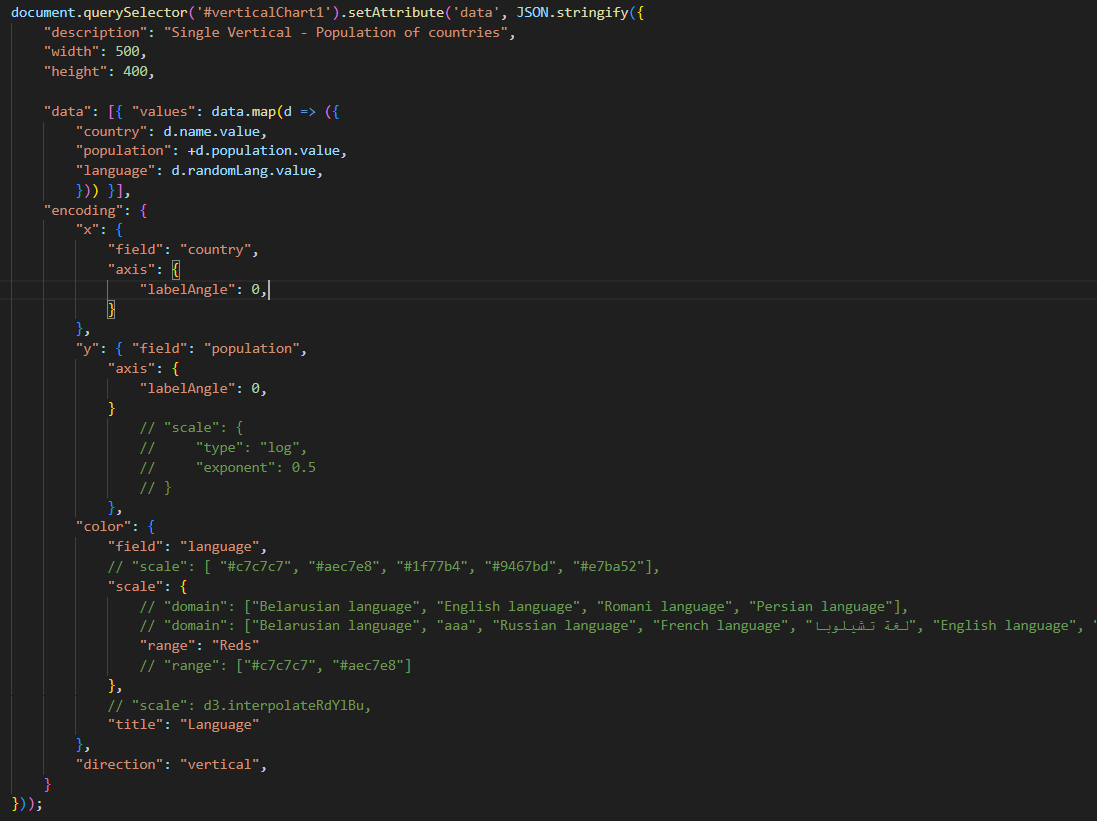
Up to now, my project has achieved some results as follows:

* Completed component bar chart including 4 different types of bar chart, direction of bar chart (vertical or horizontal) and connected component with metadata according to Vega-Lite grammar.
* Completed 2 types of component map (choropleth map and connection map) and connected them with metadata according to Vega-Lite grammar. The remaining 2 types are bubble map and hexbin map completed drawing, are being integrated into Vega-Lite grammar.
* Completed the pie chart component including drawing and integration with Vega-Lite grammar.

## 3.1 Bar Chart

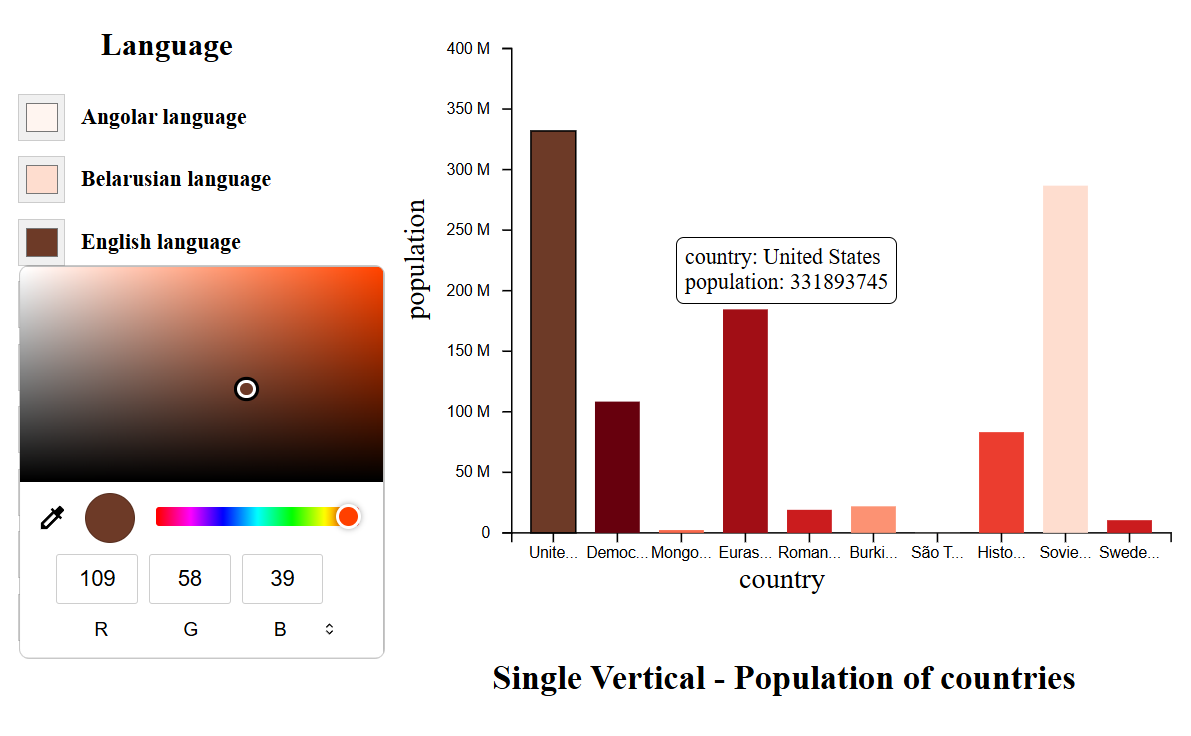
### 3.1.1 Regular Bar Chart

* Applicable when metadata includes two mandatory parameters "x" and "y".
* Each value of the variable "x" is represented by a single column.
* The height or width of the column corresponds to the "y" value.
* The color value of the "color" parameter can be set to color the axis.
* Allow axis label rotation via axis.labelAngle in encoding.
* Allows users to choose the display method for the quantitative axis as ScaleLinear, ScalePow or ScaleLog.
* Supports both vertical and horizontal charts via the direction parameter in encoding (vertical or horizontal)



*Figure 3.1 Metadata of Regular Bar Chart*

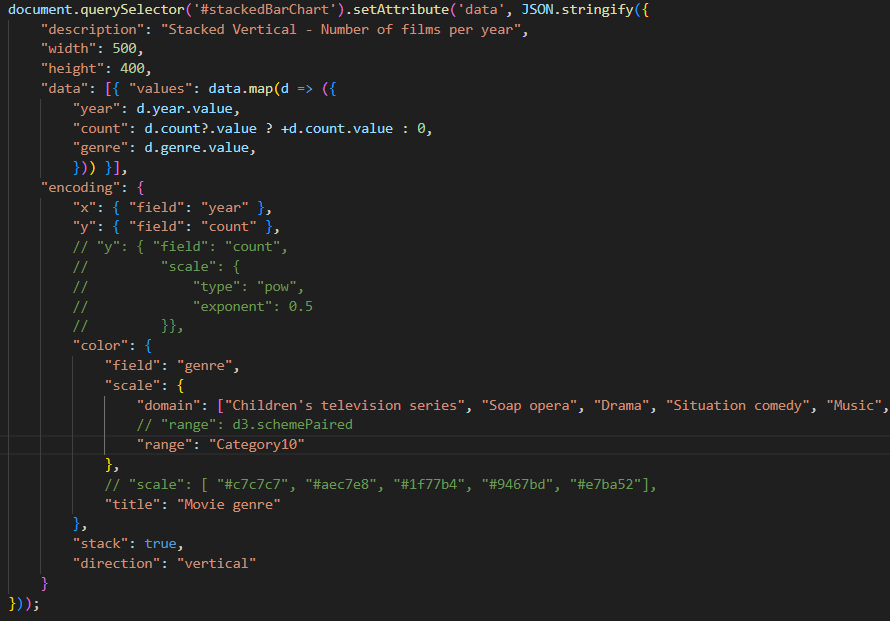
* Provide tooltip for the chart, when hovering on each part, the corresponding parameters will appear, which are the values ​​of "x" and "y".
* Allow users to change the color of "color" directly on the legend.



*Figure 3.2 Visualization of Regular Bar Chart*

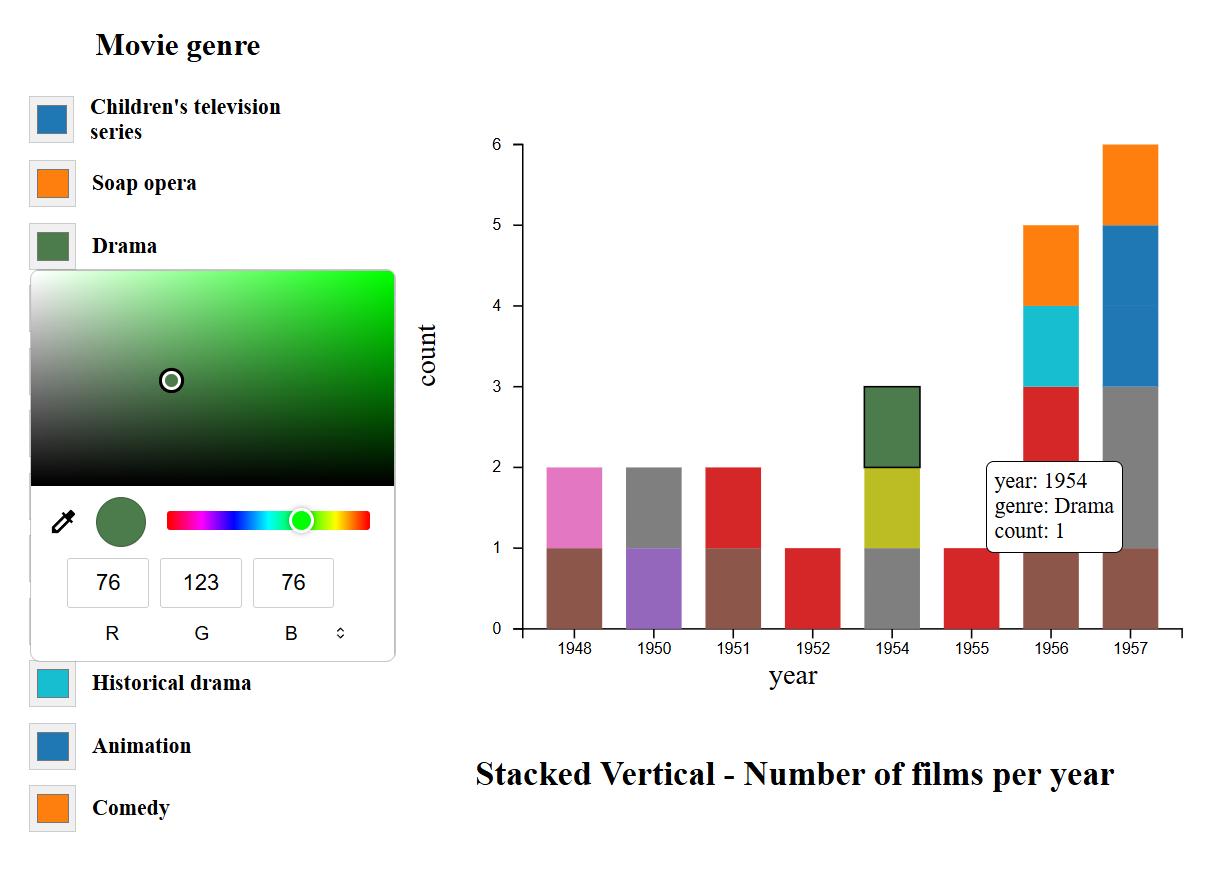
### 3.1.2 Stacked Bar Chart

* Includes 3 required parameters: "x", "y" and "color". Activated when the "stack" parameter in the encoding is true (stack: true).
* Each "x" is a summary column, stacked with different "color" elements along the "y" axis.
* Use d3.stack() to calculate the stacking order and the bottom/top values ​​of each section.
* Allows users to choose the display method for the quantitative axis as ScaleLinear, ScallePow or ScaleLog.
* Supports both vertical and horizontal charts via the "direction" parameter in encoding (vertical or horizontal).



*Figure 3.3 Metadata of Stacked Bar Chart*

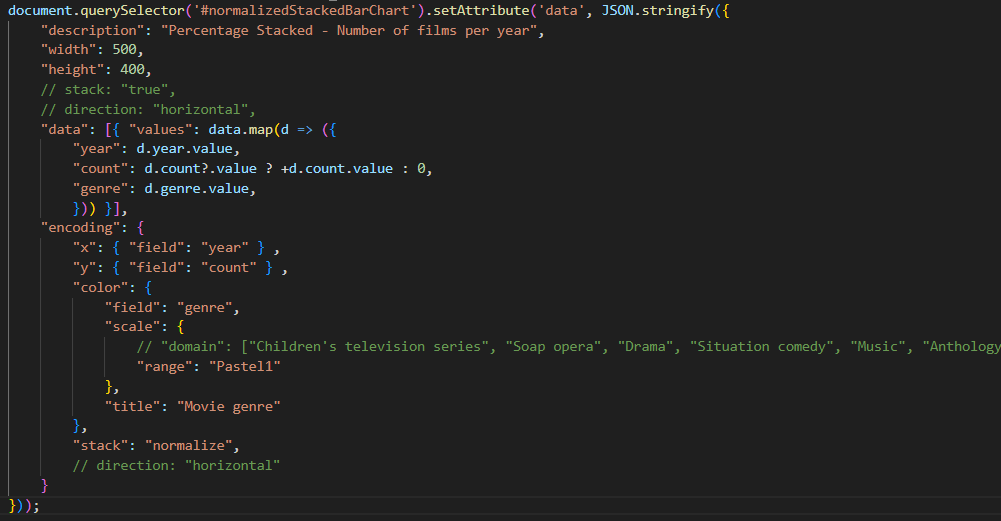
* Provide tooltip for the chart, when hovering on each part, the corresponding parameters will appear, which are the values ​​of "x", " color " and "y".
* Allow users to change the color of "color" directly on the legend.



*Figure 3.4 Visualization of Stacked Bar Chart*

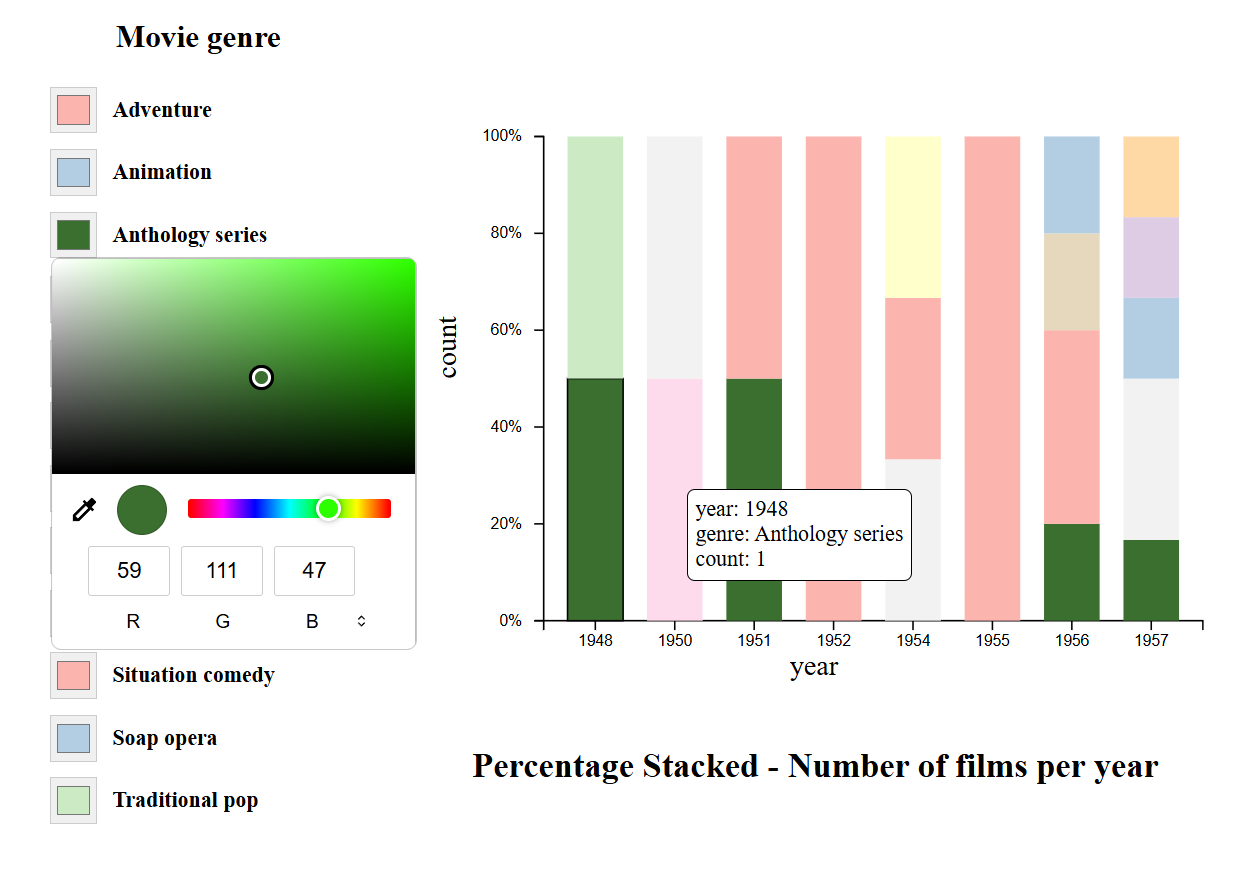
### 3.1.3 Percentage Stacked Bar Chart

* Similar to Stacked Bar Chart, the difference is that the "stack" parameter in the encoding is normalize (stack: "normalized").
* Each "x" is a summary column, stacked with different "color" elements along the "y" axis. The elements in each column are normalized to sum = 1 (100%).
* Allows for percentage comparison between groups, regardless of the total value.
* Allows users to choose the display method for the quantitative axis as ScaleLinear, ScallePow or ScaleLog.
* Supports both vertical and horizontal charts via the "direction" parameter in encoding (vertical or horizontal).



*Figure 3.5 Metadata of Percentage Stacked Bar Chart*

* Provide tooltip for the chart, when hovering on each part, the corresponding parameters will appear, which are the values ​​of "x", " color " and "y".
* Allow users to change the color of "color" directly on the legend.



*Figure 3.6 Visualization of Percentage Stacked Bar Chart*

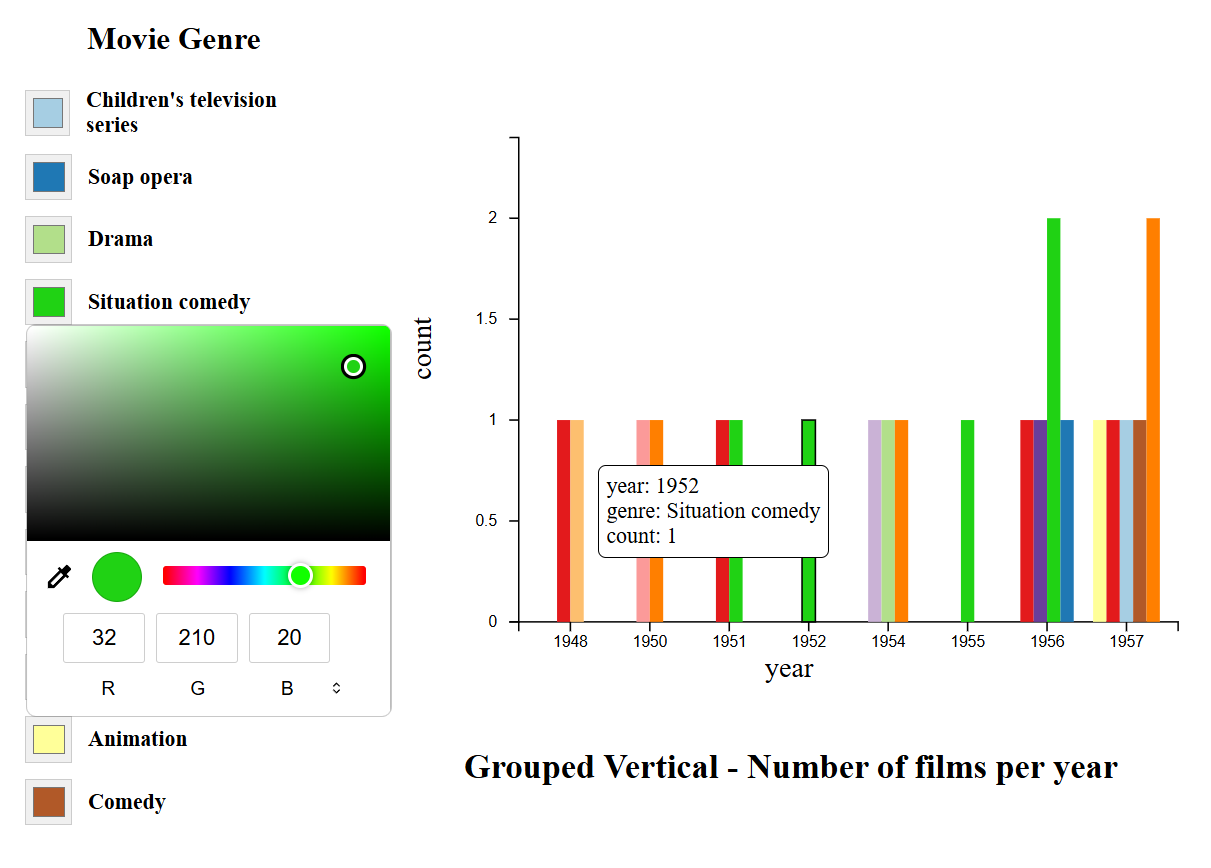
### 3.1.4 Grouped Bar Chart

* Activated when the "color" parameter in the encoding exists and stack parameter in the encoding is false (stack: false).
* Each "x" value is a group of sub-bars, each "color" represents a separate color group.
* Allows users to choose the display method for the quantitative axis as ScaleLinear, ScallePow or ScaleLog.
* Supports both vertical and horizontal charts via the "direction" parameter in encoding (vertical or horizontal).



*Figure 3.7 Metadata of Grouped Bar Chart*

* Provide tooltip for the chart, when hovering on each part, the corresponding parameters will appear, which are the values ​​of "x", " color " and "y".
* Allow users to change the color of "color" directly on the legend.



*Figure 3.8 Visualization of Grouped Bar Chart*

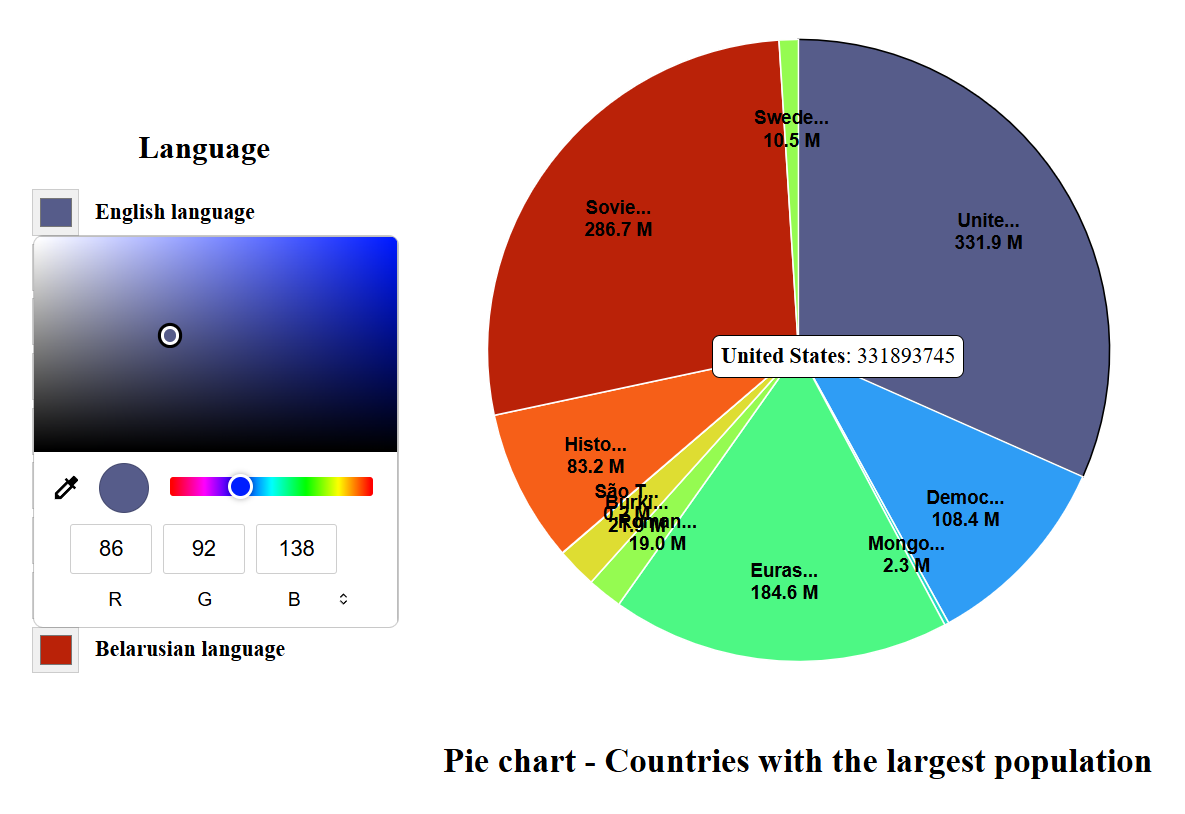
## 3.2 Pie Chart

* The pie chart requires three parameters: text, theta, and color.
* "text" is the label that corresponds to the name of each section in the chart.
* "theta" will be the value that corresponds to the section to draw the angles.
* "color" will be used to color each section of the chart.



*Figure 3.9 Metadata of Pie Chart*

* Provide tooltip for the chart, when hovering on each part, the corresponding parameters will appear, which are the values ​​of "label", and "theta".
* Allow users to change the color of "color" directly on the legend.

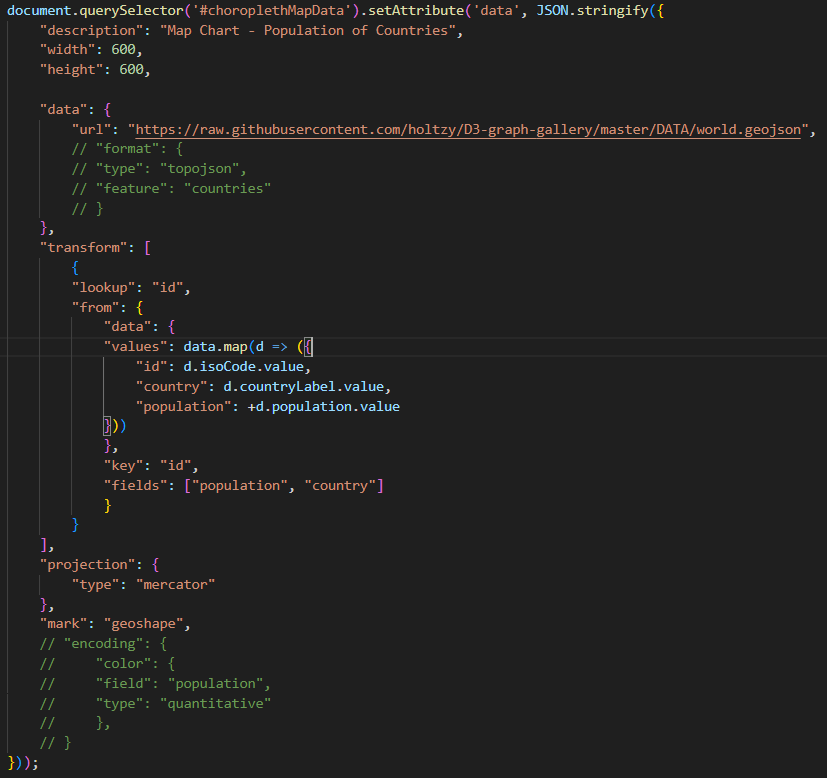


*Figure 3.10 Visualization of Pie Chart*

## 3.3 Geographic Map

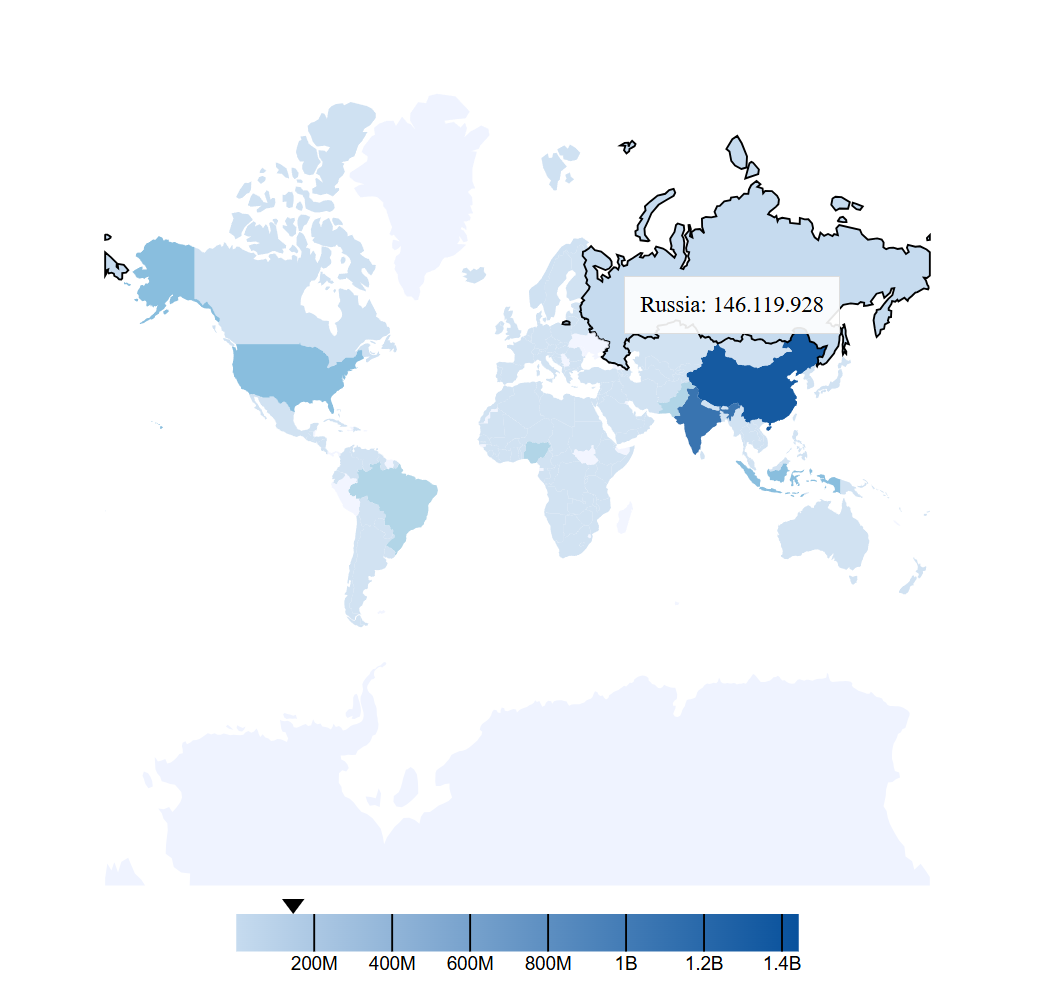
### 3.3.1 Choropleth Map

* Choropleth map requires main parameters which are map data, id to connect between map and data and fields needed to display.
* Parameter "url" in data to get GeoJSON of map.
* Parameter "lookup" in transform to get column name to connect to GeoJSON.
* Parameter "key" in "from" of transform to get data connection key and "fields" to get necessary fields.
* Projection is used to represent the type of map projection.



*Figure 3.11 Metadata of Choropleth Map*

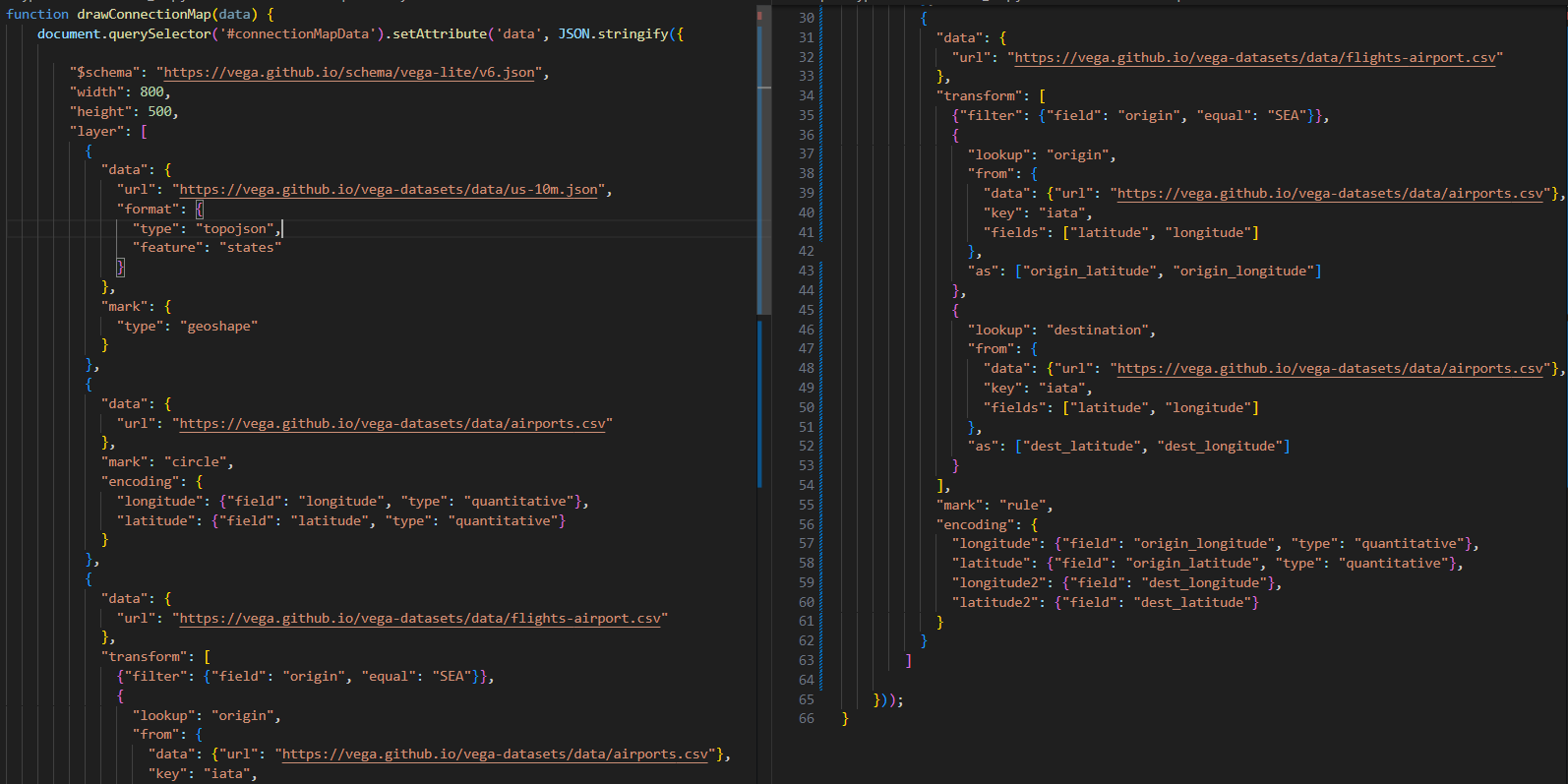
* Provide tooltip for the chart, when hovering on each part, the corresponding parameter will appear, which is the name of the position according to "id" and the value we get corresponding to that "id" in the transform section.
* The legend part will show the value of the field we get in the transform part corresponding to the currently selected part.



*Figure 3.12 Visualization of Choropleth Map*

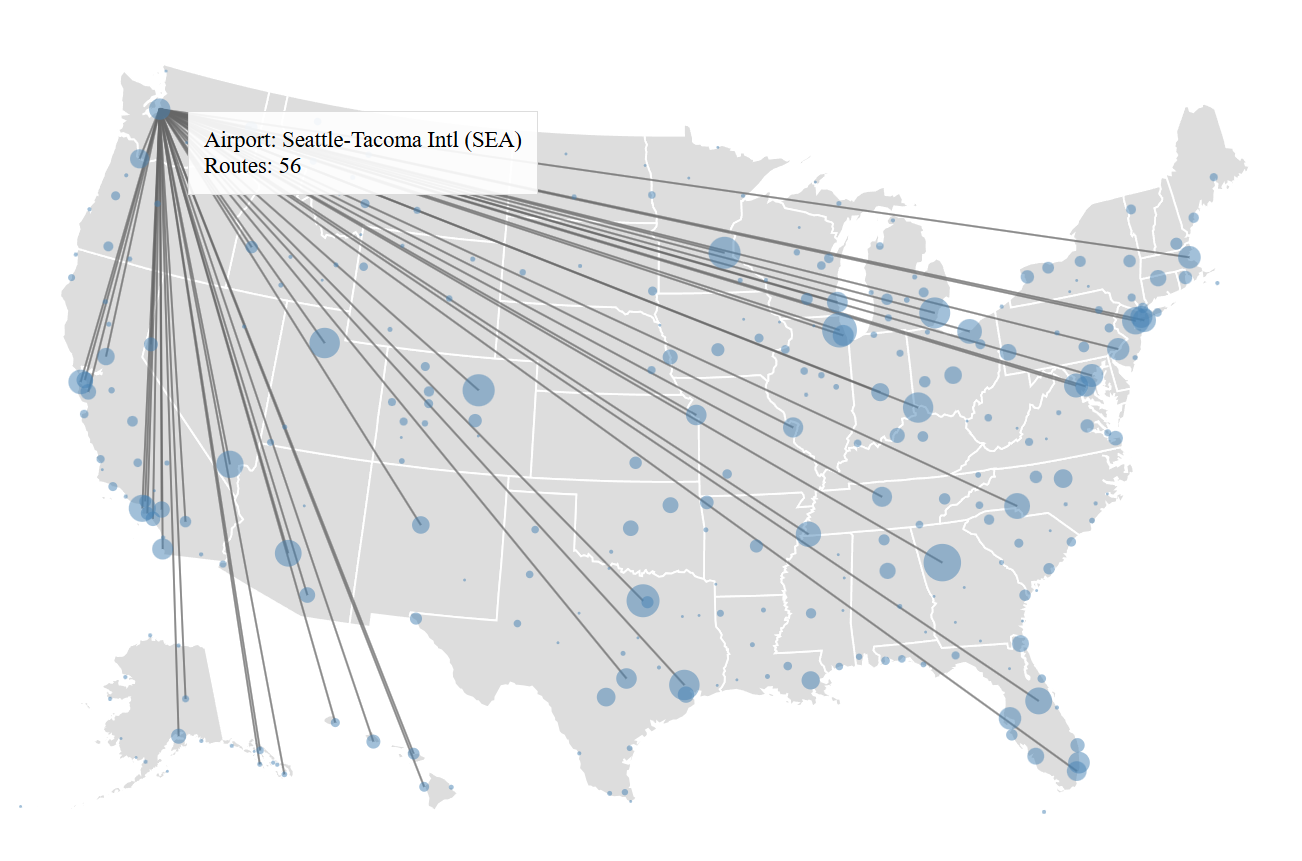
### 3.3.2 Connection Map

* ConnectionMap uses a layer structure consisting of 3 overlapping layers. Layer 1 is used to draw the map (us-10m.json), layer 2 is used to draw nodes (airport.csv) and layer 3 is used to draw links (flights-airport.csv).
* The layers will have the parameter "url" to identify the data file.
* Parameter "lookup" in transform to get column name to connect to GeoJSON.
* In the node and link layers, there will be variables "longitude" and "latitude" to identify the location, the nodes and links will be connected through the "key" in the "from" of the transform of link’s layer ("key": "iata").



*Figure 3.13 Metadata of Connection Map*

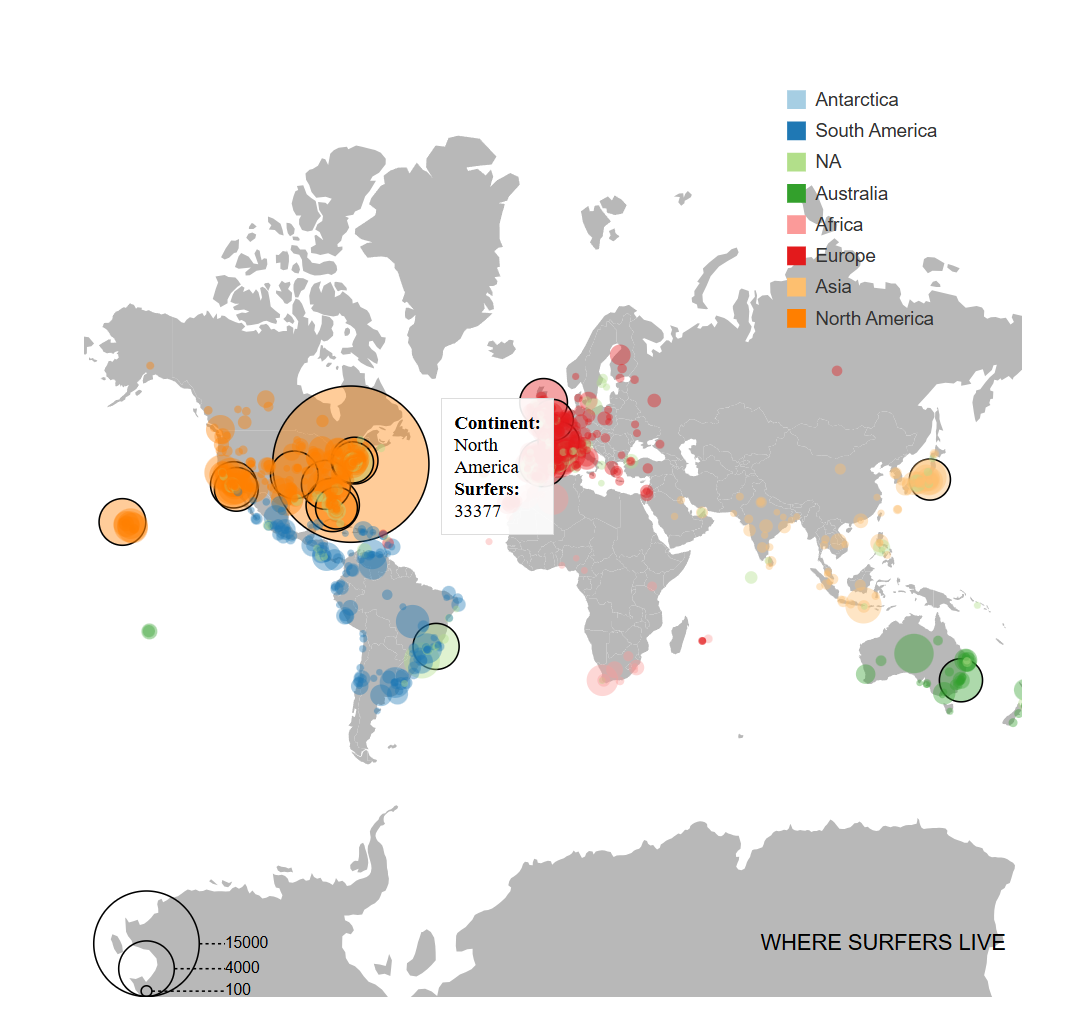
* Provide tooltip for the chart, when hovering on each part, the corresponding parameter will appear, which is the name of the node according to "id" and the link to that node through layer 3.



*Figure 3.14 Visualization of Connection Map*

### 3.3.3 Bubble Map

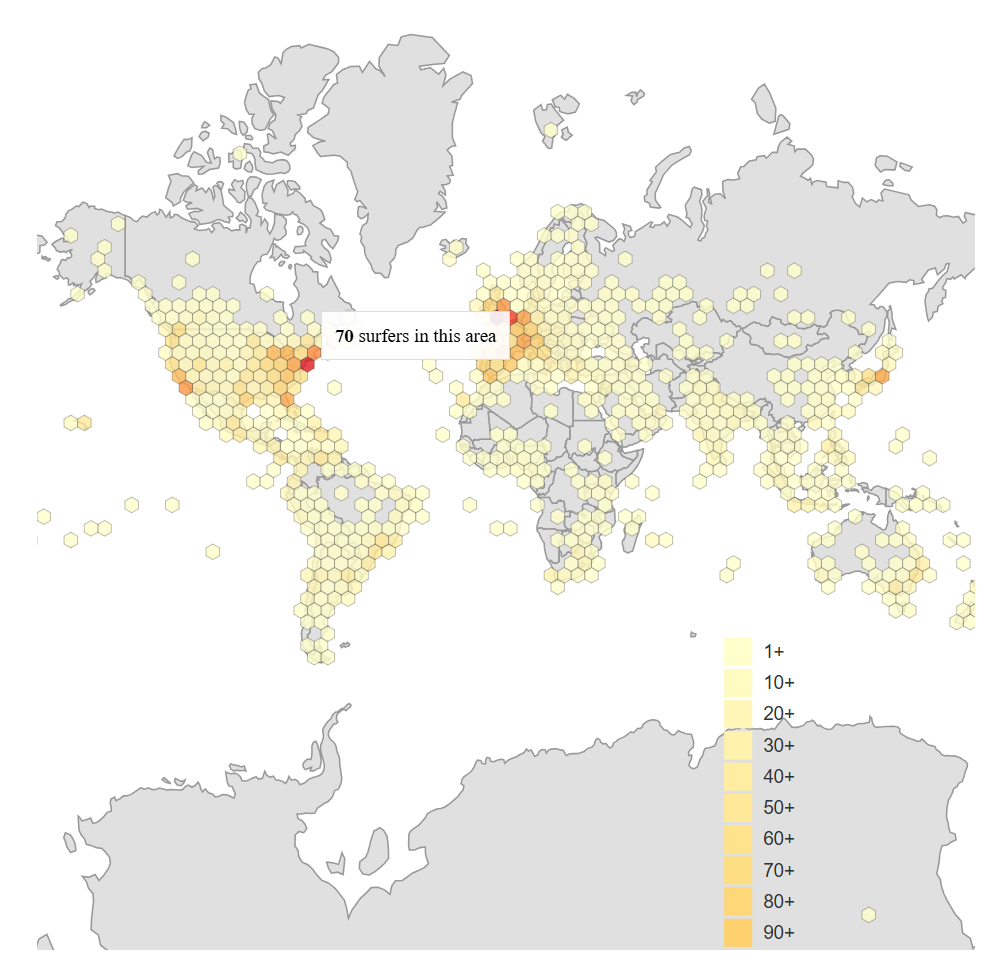
* In BubbleMap part there is currently no metadata connection, and the input data is hard-coded.
* It is only recognized through the parameter "bubble": "true".
* Includes 2 parts: GeoJSON file to draw the map and csv file to draw bubbles. GeoJSON file will be declared at the parameter "url" of data and csv file will be declared at a certain parameter and this csv file must have parameters such as longitude, latitude to indicate coordinates and n to indicate quantity.
* Provide tooltip for the chart, when hovering on each bubble, the corresponding parameter will appear, which is the number of csv file.



*Figure 3.15 Visualization of Bubble Map*

### 3.3.4 Hexbin Map

* HexbinMap is the same as BubbleMap, in HexbinMap part there is currently no metadata connection, and the input data is hard-coded.
* It is only recognized through the parameter "hexbin": "true".
* Includes 2 parts: GeoJSON file to draw the map and csv file to draw hexbin. GeoJSON file will be declared at the parameter "url" of data and csv file will be declared at a certain parameter and this csv file must have parameters such as longitude, latitude to indicate coordinates and n to indicate quantity.
* Provide tooltip for the chart, when hovering on each bubble, the corresponding parameter will appear, which is the number of csv file. The legend part will show the color corresponding to the magnitude of the value.



*Figure 3.16 Visualization of Hexbin Map*

## 3.4 Color Palettes

To help users specify color palettes flexibly when configuring charts, the library has built a function that handles color palette names in a concise and easy-to-understand syntax, allowing users to pass in strings like "Reds[5]", "Category10", or "Reds" to accurately map to available color palettes in the D3.js library like d3.schemeReds[5], d3.schemeCategory10, or d3.interpolateReds.

### 3.4.1 Purpose

* Allow users to specify color palettes using a simple, natural-language syntax.
* Supports both discrete color palettes (d3.scheme\*) and continuous color palettes (d3.interpolate\*).
* Automatically normalize color palette names to ensure compatibility with actual keys in the D3 library.

### 3.4.2 Operating principle

* Parse the input string:
  + Use a regular expression to extract the palette name and index (if any).
  + For example: "Reds[5]" → name: Reds, index: 5, "Category10" → name: Category10, no index.
* Normalize the palette name:
  + Search for keys in d3 that are in the form scheme\* or interpolate\*.
  + Compare the name after the prefix (e.g. Reds, Category10) with the user-entered name (case-insensitive).
  + Return the full key if found.
* Get the value from D3:
  + If index is present: return d3.schemeXXX[index] if valid.
  + If index is absent: use d3.interpolateXXX if present.
  + If there is no interpolate function, try returning d3.schemeXXX as an array of colors.

### 3.4.3 Syntax

Here are the inputs and the corresponding color arrays D3 returns:

### 3.4.3.1 Categorical schemes

* Category10 (= d3.schemeCategory10)



* Accent (= d3.schemeAccent)



* Dark2 (= d3.schemeDark2)



* Observable10 (= d3.schemeObservable10)



* Paired (= d3.schemePaired)



* Pastel1 (= d3.schemePastel1)



* Pastel2 (= d3.schemePastel2)



* Set1 (= d3.schemeSet1)



* Set2 (= d3.schemeSet2)



* Set3 (= d3.schemeSet3)



* Tableau10 (= d3.schemeTableau10)



### 3.4.3.2 Cyclical schemes

* Rainbow (= d3.interpolateRainbow)



* Sinebow (= d3.interpolateSinebow)



### 3.4.3.3 Diverging schemes

* BrBG (= d3.interpolateBrBG)



* PRGn (= d3.interpolatePRGn)



* PiYG (= d3.interpolatePiYG)



* PuOr (= d3.interpolatePuOr)



* RdBu (= d3.interpolateRdBu)



* RdGy (= d3.interpolateRdGy)



* RdYlBu (= d3.interpolateRdYlBu)



* RdYlGn (= d3.interpolateRdYlGn)



* Spectral (= d3.interpolateSpectral)



* BrBG[k] (= d3.schemeBrBG[k]).  *k in 3 – 11.*



* PRGn[k] (= d3.schemePRGn[k]).  *k in 3 – 11.*



* PiYG[k] (= d3.schemePiYG[k]).  *k in 3 – 11.*



* PuOr[k] (= d3.schemePuOr[k]).  *k in 3 – 11.*



* RdBu[k] (= d3.schemeRdBu[k]).  *k in 3 – 11.*



* RdGy[k] (= d3.schemeRdGy[k]).  *k in 3 – 11.*



* RdYlBu[k] (= d3.schemeRdYlBu[k]).  *k in 3 – 11.*



* RdYlGn[k] (= d3.schemeRdYlGn[k]).  *k in 3 – 11.*



* Spectral[k] (= d3.schemeSpectral[k]).  *k in 3 – 11.*



### 3.4.3.4 Sequential schemes

* Blues (= d3.interpolateBlues)



* Greens (= d3.interpolateGreens)



* Greys (= d3.interpolateGreys)



* Oranges (= d3.interpolateOranges)



* Purples (= d3.interpolatePurples)



* Reds (= d3.interpolateReds)



* Turbo (= d3.interpolateTurbo)



* Viridis (= d3.interpolateViridis)



* Inferno (= d3.interpolateInferno)



* Magma (= d3.interpolateMagma)



* Plasma (= d3.interpolatePlasma)



* Cividis (= d3.interpolateCividis)



* Warm (= d3.interpolateWarm)



* Cool (= d3.interpolateCool)



* CubehelixDefault (= d3.interpolateCubehelixDefault)



* BuGn (= d3.interpolateBuGn)



* BuPu (= d3.interpolateBuPu)



* GnBu (= d3.interpolateGnBu)



* OrRd (= d3.interpolateOrRd)



* PuBuGn (= d3.interpolatePuBuGn)



* PuBu (= d3.interpolatePuBu)



* PuRd (= d3.interpolatePuRd)



* RdPu (= d3.interpolateRdPu)



* YlGnBu (= d3.interpolateYlGnBu)



* YlGn (= d3.interpolateYlGn)



* YlOrBr (= d3.interpolateYlOrBr)



* YlOrRd (= d3.interpolateYlOrRd)



* Blues[k] (= d3.schemeBlues[k]).  *k in 3 – 11.*



* Greens[k] (= d3.schemeGreens[k]).  *k in 3 – 11.*



* Greys[k] (= d3.schemeGreys[k]).  *k in 3 – 11.*



* Oranges[k] (= d3.schemeOranges[k]).  *k in 3 – 11.*



* Purples[k] (= d3.schemePurples[k]).  *k in 3 – 11.*



* Reds[k] (= d3.schemeReds[k]).  *k in 3 – 11.*



* BuGn[k] (= d3.schemeBuGn[k]).  *k in 3 – 11.*



* BuPu[k] (= d3.schemeBuPu[k]).  *k in 3 – 11.*



* GnBu[k] (= d3.schemeGnBu[k]).  *k in 3 – 11.*



* OrRd[k] (= d3.schemeOrRd[k]).  *k in 3 – 11.*



* PuBuGn[k] (= d3.schemePuBuGn[k]).  *k in 3 – 11.*



* PuBu [k] (= d3.schemePuBu[k]).  *k in 3 – 11.*



* PuRd [k] (= d3.schemePuRd[k]).  *k in 3 – 11.*



* RdPu [k] (= d3.schemeRdPu[k]).  *k in 3 – 11.*



* YlGnBu[k] (= d3.schemeYlGnBu[k]).  *k in 3 – 11.*



* YlGn[k] (= d3.schemeYlGn[k]).  *k in 3 – 11.*



* YlOrBr[k] (= d3.schemeYlOrBr[k]).  *k in 3 – 11.*



* YlOrRd[k] (= d3.schemeYlOrRd[k]).  *k in 3 – 11.*



# Plan for Implementation

|  |  |  |
| --- | --- | --- |
|  | **Tasks** | **Timeline** |
| 1 | * Test the bar chart component and pie chart component, publish them to npm to get user feedback. * Complete Vega-Lite grammar integration for the remaining two map types, bubbleMap and HexbinMap. | 1 week  (13/05 – 20/05) |
| 2 | * Create test cases and test the map component, publish it to npm to get user reviews. * Refactor code of map component to remove duplication and reduce size. * Start working on the nodelink diagram component. * Integrate Vega-Lite grammar for nodelink diagram. | 1 week  (20/05 – 27/05) |
| 3 | * Refactor code of nodelink diagram component to remove duplication and reduce size. * Complete the nodelink component and start testing it, publish it to npm to get user ratings. * Start editing and finalizing the bar chart and pie chart components based on user ratings. | 1 week  (27/05 – 03/06) |
| 4 | * Start editing and finalizing the map and nodelink diagram components based on user ratings. * Start writing reports | 1 week  (03/06 - 10/06) |
| 5 | * Final check of the library. * Write documentation for project. | 2 week  (10/06 - 24/06) |

*Table 4.1 Tasks to do in M3*

# References

|  |  |
| --- | --- |
| [1] | "Semanctic Web Standards - Resouce Description Framework," [Online]. Available: https://www.w3.org/RDF/. |
| [2] | "W3C - Linked Open Data," [Online]. Available: https://www.w3.org/egov/wiki/Linked\_Open\_Data. |
| [3] | "W3C - SPARQL 1.1 Query Language," [Online]. Available: https://www.w3.org/TR/sparql11-query/. |
| [4] | "Echart," [Online]. Available: https://echarts.apache.org/en/index.html. |
| [5] | Menin, A., Cava, R., Dal Sasso Freitas, C. M., Corby, O., & Winckler, M., *Towards a Visual Approach for Representing Analytical Provenance in Exploration Processes.,* 05-09 July 2021. |
| [6] | "D3js," [Online]. Available: https://d3js.org/. |
| [7] | "Vega-Lite - A grammar of Interactive Graphics," [Online]. Available: https://vega.github.io/vega-lite/. |
| [8] | "MDN Web Component," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web\_components. |
| [9] | "Stencil JS," [Online]. Available: https://stenciljs.com/. |
| [10] | "NPM JS," [Online]. Available: https://www.npmjs.com/. |