THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF ADVANCED SCIENCE AND TECHNOLOGY

CAPSTONE PROJECT

DESIGN AND DEVELOPMENT OF A DATA VISUALIZATION LIBRARY FOR SPARQL QUERIES

Student: **MINH HUY DO - 123200103**
Class: **20PFIEV3**

Academic advisor: **Dr. THI MY HANH LE**
Supervisor: **Assoc. Prof., Dr. ALINE MENIN**
Co-Supervisor: **Prof., Dr. CATHERINE FARON**

Danang, 06 / 2025

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Số: 883 /QĐ-ĐHBK

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

Đà Nẵng, ngày 03 tháng 3 năm 2025

QUYẾT ĐỊNH

Về việc thành lập Hội đồng hướng dẫn Capstone project cho sinh viên ngành Công nghệ thông tin - chuyên ngành Công nghệ phần mềm, Khoa Khoa học Công nghệ tiên tiến

HIỆU TRƯỞNG TRƯỜNG ĐẠI HỌC BÁCH KHOA

Căn cứ Nghị định số 32/CP ngày 04 tháng 4 năm 1994 của Chính phủ về việc thành lập Đại học Đà Nẵng;

Căn cứ Thông tư số 10/2020/TT-BGDĐT ngày 14 tháng 5 năm 2020 của Bộ trưởng Bộ Giáo dục và Đào tạo ban hành Quy chế tổ chức và hoạt động của đại học vùng và các cơ sở giáo dục đại học thành viên;

Căn cứ Nghị quyết số 08/NQ-HĐĐH ngày 12 tháng 7 năm 2021 của Hội đồng Đại học Đà Nẵng về việc ban hành Quy chế tổ chức và hoạt động của Đại học Đà Nẵng và Nghị quyết 13/NQ-HĐĐH ngày 07 tháng 9 năm 2021 của Hội đồng Đại học Đà Nẵng về việc sửa đổi, bổ sung một số điều của Quy chế tổ chức và hoạt động của Đại học Đà Nẵng;

Căn cứ Nghị quyết số 93/NQ-HĐT ngày 18 tháng 12 năm 2023 của Hội đồng trường Trường Đại học Bách khoa ban hành Quy chế tổ chức và hoạt động của Trường Đại học Bách khoa, Đại học Đà Nẵng;

Căn cứ Quyết định số 840/QĐ-ĐHBK ngày 28 tháng 02 năm 2023 của Hiệu trưởng Trường Đại học Bách khoa, Đại học Đà Nẵng ban hành Quy định về việc tổ chức và triển khai thực hiện Thực tập tốt nghiệp và Đồ án tốt nghiệp theo hình thức Capstone Project của Trường Đại học Bách khoa, Đại học Đà Nẵng;

Căn cứ Biên bản cuộc họp ngày 21 tháng 02 năm 2025 về việc Họp xét giao Đồ án tốt nghiệp học kỳ 2, năm học 2024-2025;

Căn cứ Tờ trình số 30/TTr-KHCNTT ngày 28 tháng 02 năm 2025 của Trưởng khoa Khoa Khoa học Công nghệ tiên tiến về việc giao đề tài Capstone project cho sinh viên ngành Công nghệ thông tin - chuyên ngành Công nghệ phần mềm, Khoa Khoa học Công nghệ tiên tiến;

Theo đề nghị của Trưởng phòng Phòng Đào tạo.

QUYẾT ĐỊNH:

Điều 1. Thành lập Hội đồng hướng dẫn Capstone project cho 29 sinh viên khóa 2020 và 2019 trở về trước, Chương trình đào tạo kỹ sư chất lượng cao Việt - Pháp, ngành Công nghệ thông tin - chuyên ngành Công nghệ phần mềm, Khoa Khoa học Công nghệ tiên tiến. Tên đề tài, sinh viên thực hiện và danh sách Hội đồng hướng dẫn được đính kèm theo Quyết định.

Điều 2. Hội đồng hướng dẫn tổ chức cho sinh viên thực hiện Capstone project theo quy định hiện hành. Sau khi hoàn thành nhiệm vụ, Hội đồng tự giải thể.

Điều 3. Trưởng phòng Phòng Tổ chức - Hành chính, Trưởng phòng Phòng Đào tạo, Trưởng phòng Phòng Kế hoạch - Tài chính, Trưởng khoa Khoa Khoa học Công nghệ tiên tiến và viên chức, sinh viên có tên ở Điều 1 căn cứ Quyết định thi hành./.

Nơi nhận:
- Như Điều 3;
- Khoa Khoa học Công nghệ tiên tiến;
- Lưu: VT, ĐT.

KT. HIỆU TRƯỞNG
PHÓ HIỆU TRƯỞNG

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC BÁCH KHOA
ĐẠI HỌC ĐÀ NẴNG

PGS.TS. Nguyễn Hồng Hải

ĐỀ TÀI 5			
1.Tên đề tài: **Design and development of a data visualization library for SPARQL queries (Thiết kế và phát triển thư viện trực quan hóa dữ liệu cho các truy vấn SPARQL).** (Đề tài thực hiện và bảo vệ bằng Tiếng Anh)			
2. Sinh viên thực hiện	Đỗ Minh Huy	20PFIEV3	
3. Hội đồng hướng dẫn	TS. Lê Thị Mỹ Hạnh	Giảng viên, Khoa Công nghệ thông tin, Trường Đại học Bách khoa	Chủ tịch – Hướng dẫn
	Mời GS. Aline Menin	Laboratoire I3S, Campus SophiaTech - Inria Sophia Antipolis, Fance	Đồng hướng dẫn

SUPERVISOR'S COMMENT

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

…….

UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF ADVANCED SCIENCE AND TECHNOLOGY

SOCIALIST REPUBLIC OF VIETNAM
Independence - Freedom - Happiness

GRADUATION PROJECT TASKS
NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

I. General information (Thông tin chung):

- Full name of student (Họ tên sinh viên): Minh Huy Do Class (Lớp): 20PFIEV3
- Student ID (Mã số sinh viên): 123200103

II. Project overview (Tổng quan về đồ án):

1. Project's title (Tên đề tài): Design and development of a data visualization library for SPARQL queries (Thiết kế và phát triển thư viện trực quan hoá dữ liệu cho các truy vấn SPARQL)
2. Contents of the report sections (Nội dung các phần của báo cáo):
    - Chapter 1: Review of literature (Chương 1: Tổng quan tài liệu)
    - Chapter 2: Detailed theory (Chương 2: Lý thuyết chi tiết)
    - Chapter 3: Proposed technology/system (Chương 3: Công nghệ/hệ thống đề xuất)
    - Chapter 4: Obtained results (Chương 4: Kết quả thu được)
3. Full name of academic advisor (Họ tên của Giáo viên hướng dẫn): Dr., Thi My Hanh LE
4. Full name of supervisor (Họ tên của người hướng dẫn): Assoc. Prof., Dr. Aline Menin
5. Project assignment date (Ngày giao nhiệm vụ đồ án): 01/02/2025
6. Project completion date (Ngày hoàn thành đồ án): 17/06/2025

Da nang, date month ,2025
Full name & signature
of the Chairman of the Advisory Board
(Họ tên & chữ ký Chủ tịch Hội đồng hướng dẫn)

ABSTRACT

In the context of the ever-growing volume of RDF (Resource Description Framework) datasets published as Linked Open Data (LOD), the need for accessible and effective data visualization has become critical. While SPARQL offers a powerful querying mechanism for such data, the challenge remains in presenting query results in a comprehensible, interactive, and reusable visual format—especially for users without extensive programming knowledge. Most existing tools are limited to CSV/JSON datasets and do not cater well to the structural complexity of RDF.

To address this gap, a JavaScript-based visualization library was developed to support semantic web practitioners and decision-makers in transforming SPARQL query results into meaningful visual representations. The system integrates Vega-Lite grammar for high-level specification of charts and utilizes Web Components to ensure modularity, reusability, and performance across web platforms. The library supports a variety of visual formats including bar charts (regular, stacked, grouped, and percentage), pie charts, and geographical maps (choropleth, connection, bubble, and hexbin), all configurable through a JSON-based metadata layer.

Up to now, several chart types have been successfully implemented, fully integrated with SPARQL outputs and Vega-Lite configurations. Additionally, a flexible color palette handler was introduced to allow users to define D3.js-compatible color schemes in a simplified syntax. These results demonstrate the potential for a reusable, scalable solution for semantic data visualization, significantly lowering the technical barrier for non-programmers.

ACKNOWLEDGMENT

This graduation thesis was conducted in 2025 during my internship at the WIMMICS research team, INRIA Sophia Antipolis, France. I am profoundly grateful to Associate Professor Dr. Aline MENIN for her dedicated supervision, thoughtful guidance, and continuous support throughout my time at WIMMICS. Her expertise, patience, and encouragement played a pivotal role in shaping both the direction and quality of this thesis.

I would also like to express my sincere appreciation to Dr. Thi My Hanh LE, my academic advisor at the Faculty of Information Technology, University of Science and Technology – The University of Danang. Her unwavering support, constructive feedback, and insightful suggestions greatly enriched my research experience and academic growth. I am also deeply grateful to the instructors and staff at the Faculty of Information Technology and the Faculty of Advanced Science and Technology (FAST), who provided a solid academic foundation and a nurturing research environment. Special thanks to my colleagues and classmates for their collaboration, shared knowledge, and enthusiasm that made the development process both productive and enjoyable.

I am thankful to the entire WIMMICS team for welcoming me into a highly professional and intellectually stimulating environment. Their openness, collaboration, and willingness to share knowledge have left a lasting impression and significantly contributed to the success of this work.

My deepest gratitude goes to my family and close friends for their unwavering support, encouragement, and patience throughout my studies. Their love and belief in me have been a constant source of motivation.

Finally, I would like to thank all the individuals - teachers, colleagues, and fellow students - who have supported me directly or indirectly during the course of this journey.

Sophia Antipolis, June 2025
Minh Huy Do

Contents

List of Figures

List of Table

Abbreviations

Acronym	Full Form	Explanation
RDF	Resource Description Framework	A standard model for data interchange on the Semantic Web. RDF expresses data as triples (subject, predicate, object), enabling structured and linked data representation.
SPARQL	SPARQL Protocol and RDF Query Language	A query language specifically designed for querying RDF datasets, functioning similarly to SQL but for Semantic Web data.
GUI	Graphical User Interface	A user interface that allows users to interact with the system through graphical elements like buttons, charts, and icons, rather than command-line code.
JSON	JavaScript Object Notation	A lightweight data-interchange format that is easy to read and write. It is widely used to transmit data between servers and web applications.
GeoJSON	Geographic JavaScript Object Notation	A format based on JSON that encodes geographic data structures like points, lines, and polygons. Commonly used in web mapping applications.
CSV	Comma-Separated Values	A simple file format used to store tabular data, where each line represents a row and columns are separated by commas.
API	Application Programming Interface	A set of rules and protocols that allows different software applications to communicate and exchange data. In your case, this may refer to endpoints returning RDF data.
URL	Uniform Resource Locator	A reference (web address) used to access resources on the Internet, such as datasets or GeoJSON files.
D3	Data-Driven Documents	A JavaScript library for creating dynamic, interactive data visualizations using HTML, SVG, and CSS. Used to build complex charts and maps.
SVG	Scalable Vector Graphics	An XML-based format for vector images, used in the web to display high-quality graphics that can scale without losing resolution (often used in chart rendering).

Introduction

1. Motivation

- In recent years, the demand for reusable and customizable data visualizations has increased significantly, especially in the context of Linked Data, Semantic Web, and open data applications. These visual tools not only enhance user interaction but also help users interpret complex datasets more intuitively.
- From a technical perspective, existing chart libraries often lack semantic compatibility and customization for linked data applications. While tools like Vega and Vega-Lite provide a powerful grammar for declarative chart construction, integrating them seamlessly into modular, reusable web components and publishing them as NPM packages remains a non-trivial task. This project aims to bridge that gap by developing a set of semantically meaningful, customizable, and lightweight chart components that can be reused across various web platforms and applications.

2. Contribution of the Thesis

- Development of a modular chart component library based on Vega-Lite, including Pie Chart, Bar Chart, Stacked/Grouped/Normalized Bar Charts, Choropleth Maps, Bubble Maps, Hexbin Maps, and Node-Link Diagrams.
- Implementation of a fully customizable chart configuration system, allowing users to set color palettes, axis types (linear, log, pow), direction, label rotation, and stacking behaviors through intuitive parameters.
- Support for semantic integration via SPARQL query results and RDF-based data, enhancing the components' usability in semantic web contexts.
- Publishing reusable components to NPM for community access and feedback, demonstrating practical applicability and extensibility.
- Extensive code refactoring and testing to ensure performance, scalability, and maintainability of each component.

3. Organization of the Thesis

- **Chapter 1** introduces the background of the project, covering Semantic Web foundations, web components, Vega/Vega-Lite grammar, and related technologies.
- **Chapter 2** details the design and implementation of the chart library, including key challenges, architectural choices, and integration techniques.
- **Chapter 3** presents the evaluation of the developed components, discusses testing strategies, user feedback, and performance analysis.
- **Chapter 4** concludes the thesis with reflections, limitations, and future improvement directions.

4. Work Distribution

- Studied Semantic Web, RDF, SPARQL, Vega/Vega-Lite
- Developed modular chart components and integrated Vega-Lite grammar
- Designed customizable parameters and interaction behaviors for charts
- Refactored codebase, tested components, and published NPM packages
- Wrote technical documentation and thesis report

Chapter 1 Overview

1.1. Introduction of project

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.

1.2. Existing Solutions

Currently, there are only a few papers proposing tools to support RDF data visualization and SPARQL queries. However, they all have their own limitations, which prevent them from fully meeting users' needs for data visualization in a flexible and easy-to-use way. Some popular tools and papers include:

- **Echarts:** ECharts is a powerful JavaScript library that provides a variety of interactive charts such as bar charts, line charts, network charts, etc.
  - **Advantages:** Supports many types of beautiful and highly customizable charts. Well integrated with web applications and has high performance.

Supports dynamic interactions, giving users a better experience when analyzing data.

  - **Disadvantages:** Does not directly support RDF or SPARQL, users need to process data before being able to use ECharts. No module available to visualize data in RDF graph structure.

- **D3.js:** D3.js a popular JavaScript library for dynamic data visualization. It provides the ability to create complex charts by directly manipulating the DOM and SVG.
  - **Advantages:** Provides extreme customization for different types of charts. Good support for graphical data visualization. Has a large development community and rich documentation.
  - **Disadvantages:** Requires advanced JavaScript programming knowledge to use effectively. No RDF or SPARQL support available, requires data processing before display. Creating complex visualizations can be time-consuming and laborious.
- **MGExplorer - LDViz:** MGExplorer is a tool developed by the WIMMICS [7] team and it allows exploring SPARQL results through another tool, LDViz [8], which is used to explore RDF data through a graphical interface, allowing users to search and browse data by link structure.
  - **Advantages:** Supports visualization of RDF data in graph form. Integrated with direct SPARQL query capability.
  - **Disadvantages:** LDViz allows to configurate different SPARQL queries to be visualized with MGExplorer. But, indeed, the visual mapping is limited to the existing visualization techniques.
- **VOWL (Visualization of Ontologies Web Language):** VOWL [9] is a specialized tool for visualizing ontologies based on RDF/OWL, focusing on displaying the relationships between classes and properties in an ontology.
  - **Advantages:** Provides a visual approach to understanding the structure of the ontology. Graph-based data visualization, suitable for viewing relationships between entities. Supports interaction and direct navigation in the data model.

- **Disadvantages:** Only focuses on ontology visualization, not a general tool for all RDF data. Does not support many different types of graphs other than node-link graphs. Not much customization of the display interface.

- **Some related papers :** Encodable: Configurable Grammar for Visualization Components [10] and Knowledge Graph Based Visual Search Application [11].
  - **Disadvantages:** Encodable lacks chart extensions like legends, coordinate axes, and many interactive features that visual charts need. And the Knowledge Graph only allows users to use a very limited number of charts.

Chapter 2 Detailed theory

With current tools, there are always certain limitations — some libraries offer a wide variety of chart types but lack native support for RDF/SPARQL data, while others that do support SPARQL are often restricted to a narrow set of visualization styles and lack flexibility in customization. Given these shortcomings, the proposed solution aims to create an accessible, reusable JavaScript-based library tailored specifically for SPARQL query visualization. This library is designed to lower the entry barrier for users, especially those who are not experts in data visualization, requiring only basic programming knowledge and a foundational understanding of web semantics.

To develop such a solution, it is essential to understand the core technologies that underpin the Semantic Web and interactive visualizations. This chapter presents the theoretical background necessary for building the proposed system, including Semantic Web standards like RDF and SPARQL, key data visualization principles, the Vega/Vega-Lite grammar for declarative visual specifications, and modern web development techniques such as Web Components for reusable, encapsulated UI design.

2.1. Semantic Web technologies: RDF and SPARQL

The Semantic Web is an evolution of the traditional web, where data is given structured meaning, enabling machines to process and interpret information intelligently. At the heart of this ecosystem lies the Resource Description Framework (RDF) and SPARQL, both of which play crucial roles in representing and querying structured, linked data.

- RDF: A Framework for Structured Data
  - RDF (Resource Description Framework) is a W3C standard designed to represent information about resources in the form of subject-predicate-object triples. These triples together form a graph structure, where entities (nodes) are connected through relationships (edges). For instance:
    ```
    <http://example.org/person1> foaf:name "Alice" .
    ```
  - This triple represents that the person identified by <person1> has the name "Alice", using the FOAF vocabulary. RDF data can be serialized in various formats such as Turtle, RDF/XML, JSON-LD, and N-Triples.
  - RDF is schema-flexible and highly expressive, which makes it suitable for

integrating heterogeneous data from different domains. It also supports inference through ontologies defined in RDFS (RDF Schema) or OWL (Web Ontology Language), allowing new knowledge to be derived from existing data.

- SPARQL: Querying RDF Data
  - SPARQL is the standard query language for RDF. It allows users to extract and manipulate data stored in RDF format through structured patterns. A basic SPARQL SELECT query looks like this:

```
SELECT ?author ?title WHERE {
     ?book dc:title ?title .
     ?book dc:creator ?author .
 }
```

  - This query retrieves the titles and authors of books described in RDF data. SPARQL supports a variety of features including filtering (FILTER), optional patterns (OPTIONAL), grouping (GROUP BY), and aggregation functions (COUNT, AVG, etc.).
  - SPARQL queries can be executed against SPARQL endpoints, such as: DBpedia, Wikidata, Custom endpoints,….
  - These technologies enable flexible data retrieval and serve as the foundation for building knowledge-based applications that go beyond static tabular queries.

2.2. Data Visualization Principles

Data visualization involves the graphical representation of data to help users understand patterns, trends, and insights. In the context of Semantic Web data, visualization is particularly challenging due to the graph-based and often non-uniform structure of RDF datasets.

Key visualization concepts include:

- Visual encoding: Mapping data fields to graphical attributes such as position, size, color, and shape.
- Chart types: Bar charts, line graphs, pie charts, scatter plots, and network diagrams.

- Interactivity: Zooming, filtering, highlighting, and tooltips to enhance data exploration.

For RDF data, graph visualizations can be used to depict relationships, while statistical charts are needed to summarize and aggregate query results. Therefore, a flexible, general-purpose visualization approach must support both graph-based and tabular perspectives.

2.3. Vega and Vega-Lite Grammar

Vega is a declarative language for defining, storing, and sharing interactive visualization designs in a structured, JSON-based format. It separates data, layout, encoding, and interaction, allowing users to create complex visualizations without writing imperative JavaScript code.

Vega-Lite, a high-level abstraction over Vega, simplifies common visualization tasks with fewer specifications while retaining expressive power:

- Declarative specification of data, marks, and encodings.
- Support for interactivity: Brushing, filtering, tooltips, zooming.
- Modular and extensible design.
- Compatible with multiple chart types, such as bar, line, scatter, and more.

Example:

```
{
  "data": {
    "values": [
      {"category": "A", "value": 30},
      {"category": "B", "value": 80}
    ]
  },
  "mark": "bar",
  "encoding": {
    "x": {"field": "category", "type": "nominal"},
    "y": {"field": "value", "type": "quantitative"}
  }
}
```

By integrating Vega, users can create customizable and reusable visualizations for SPARQL results.

2.4. Web Components for Visualization Systems

Web Components are a modern browser standard that allows developers to create encapsulated, reusable UI components using native web technologies. They consist of:

- Custom Elements: Define new HTML tags with custom behavior.
- Shadow DOM: Encapsulates internal styles and markup.
- HTML Templates: Enable reusable structures and logic.

Advantages:

- Framework independence: Can be used with any JavaScript framework or no framework at all.
- Encapsulation: Prevents CSS and JS conflicts.
- High performance: Lightweight and runs natively in modern browsers.
- Reusability: Components can be packaged and reused across applications.

In the proposed system, Web Components are used to wrap visualization logic, data fetching, and rendering processes. This makes it easier to distribute the tool and allows users to embed visualizations into any web page using simple HTML tags.

Chapter 3 Proposed methods

3.1 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.

3.2 System architecture:

3.2.1 Block diagram:

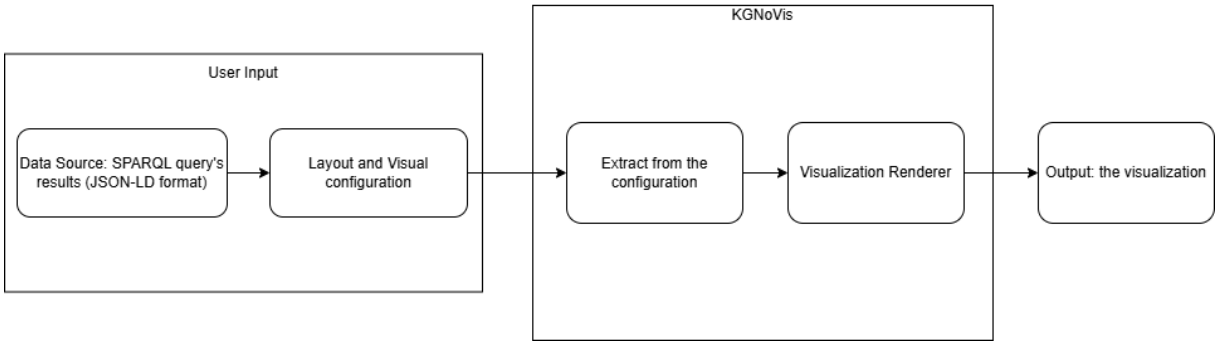

Figure 3.1: Block diagram of the system

The system is designed with a 5-step processing process:

- **User Input (SPARQL Query):** This is the entry point of the system. The system supports various data formats and endpoint protocols, making it adaptable to diverse RDF datasets. The SPARQL queries return results in standard formats (such as SPARQL JSON results), which are parsed and prepared for visualization. Users can either:
  - Enter SPARQL queries directly into the interface, which are then sent to SPARQL endpoints such as DBpedia, Wikidata, or user-defined triple stores.
  - Upload RDF files (JSON-LD format) allowing offline or preprocessed data to be used.
- **Layout and visual configuration:** Once data is available, the system transitions into a layout configuration phase where users specify how data should be visually represented. This step involves:
  - Choosing the chart type.
  - Mapping data fields to visual variables such as: « x », « y », « color »,…
  - Using Vega-Lite grammar: a declarative language that allows users to define chart behavior and appearance through JSON. This abstracts away low-level JavaScript logic while maintaining high customization.
- **Extract from the configuration:** This module parses the user-provided visual configuration and RDF data:
  - Extracts data fields from the JSON-LD structure using key-path resolution or property references.
  - Maps them to Vega-Lite encoding rules.
  - Validates that data types are compatible with selected chart types (e.g., strings for categories, numbers for axes).
- **Visualization Renderer:** Once we have the parameters from the previous extraction step, we will use it for rendering visualization. Rendered visualization is encapsulated in a Web Component, enabling easy reuse in any HTML-based system. This component:
  - Generates the Vega-Lite specification dynamically from extracted configuration and parsed RDF data.

  - Is implemented using standard Web Components (based on MDN specifications). This ensures: encapsulation, reusability and avoid dependency on external rendering frameworks.
- **Output:** Display the resulting chart in the web interface according to the input data and the selected chart with the fully interactive and responsive.

3.2.2 Activity diagram:

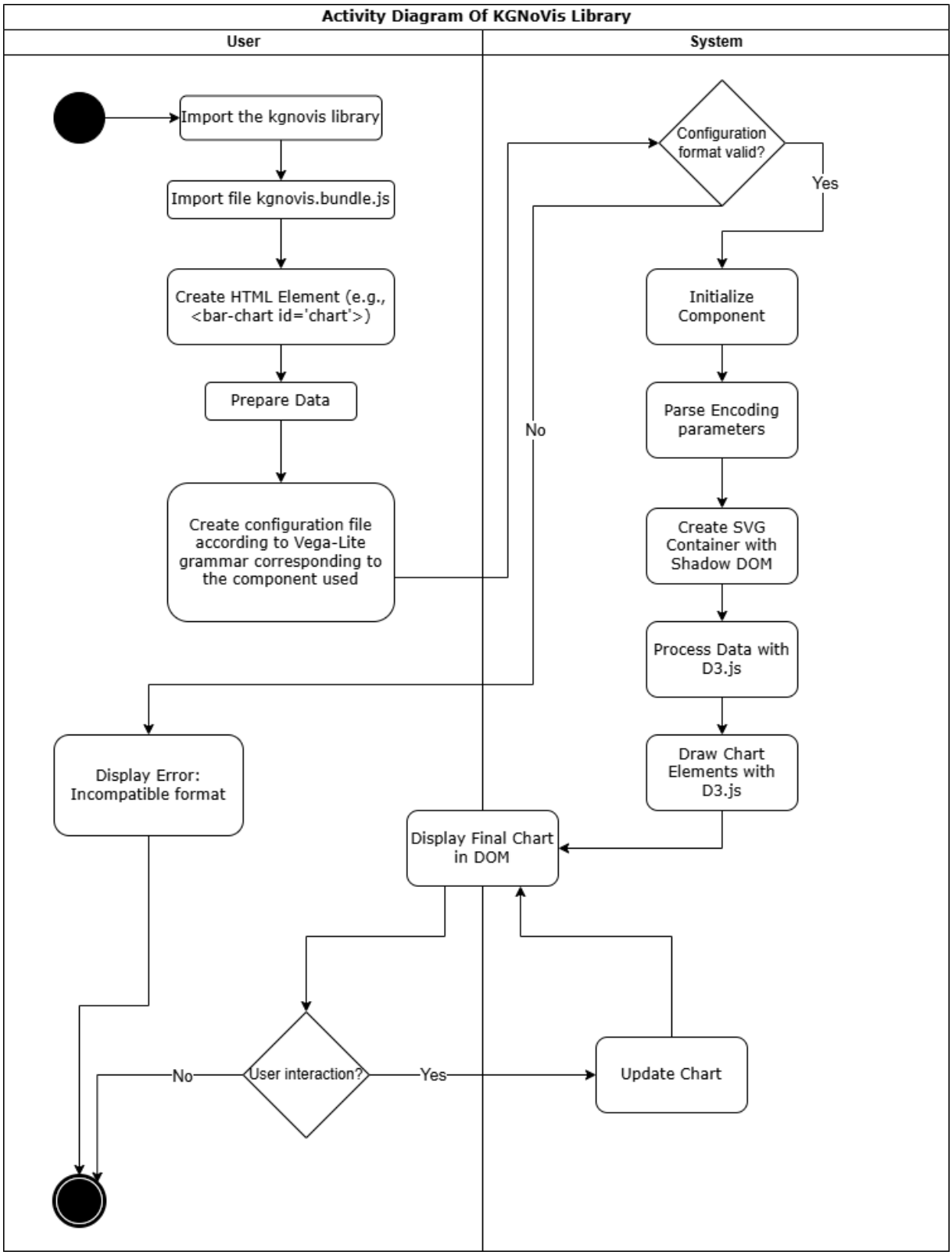

Figure 3.2 Activity diagram

This activity diagram illustrates the workflow of using the KGNoVis library to visualize RDF data through user-defined chart configurations. It represents interactions

between the User and the System (KGNoVis engine), showing each step from initialization to rendering and user interaction.

- **User-Side Activities:**
  - Import the KGNoVis library: The user starts by import the library from NPM Package to use.
  - Import the main JavaScript bundle: The user starts by importing the main JavaScript bundle (kgnovis.bundle.js) into their HTML or web application environment.
  - Create HTML Element: A chart placeholder element is created in the HTML DOM, e.g., <bar-chart id="chart1">.
  - Prepare Data: The user prepares data in JSON-LD format, typically from a SPARQL query.
  - Create Configuration file: The user defines a configuration file based on Vega-Lite grammar, which describes how the data should be visualized (e.g., axes, marks, color, size).
- **System-Side Activities:**
  - Validate Configuration: The system first checks if the configuration file is valid. If the format is incorrect, an error is returned to the user ("Incompatible format").
  - Initialize Component: If the configuration is valid, the component is initialized and DOM manipulation begins.
  - Parse Encoding Parameters: The configuration is parsed to extract visual encoding instructions (e.g., what field maps to x-axis, color, etc.).
  - Create SVG Container with Shadow DOM: The system encapsulates the visualization using Shadow DOM, isolating styles and structure.
  - Process Data with D3.js: RDF data is transformed into a visual-friendly structure using D3.js.
  - Draw Chart Elements: The chart is rendered using D3.js, based on the parsed parameters and prepared data.
  - Display Final Chart in DOM: The resulting visualization is embedded into the original HTML element.
- **User Interaction Loop:**

- Check for User Interaction: If the user interacts with the chart (e.g., filters, zoom, hover), the system detects this and updates the visualization accordingly.
- Update Chart: Based on the interaction, the chart is reprocessed and re-rendered in real time.

3.3 Required tools and materials

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

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

Chapter 4 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.

In addition to chart-specific parameters, all visualization components now support a set of common configurable parameters aligned with Vega-Lite's design principles. These shared parameters include:

- « description »: for adding semantic or explanatory notes about the chart.
- « width » and « height »: for controlling the size of the rendered chart.
- An enhanced parameter « legend » has also been introduced, which allows users to explicitly show or hide the chart legend. This provides additional flexibility for customizing the user interface based on data complexity or display preferences.

4.1 Bar Chart

To differentiate between the four types of bar charts implemented in the system - **Regular**, **Stacked**, **Percentage Stacked**, and **Grouped** - the system relies on the configuration parameter `encoding.stack` provided by the user within the Vega-Lite grammar.

The flow for determining the chart type is as follows:

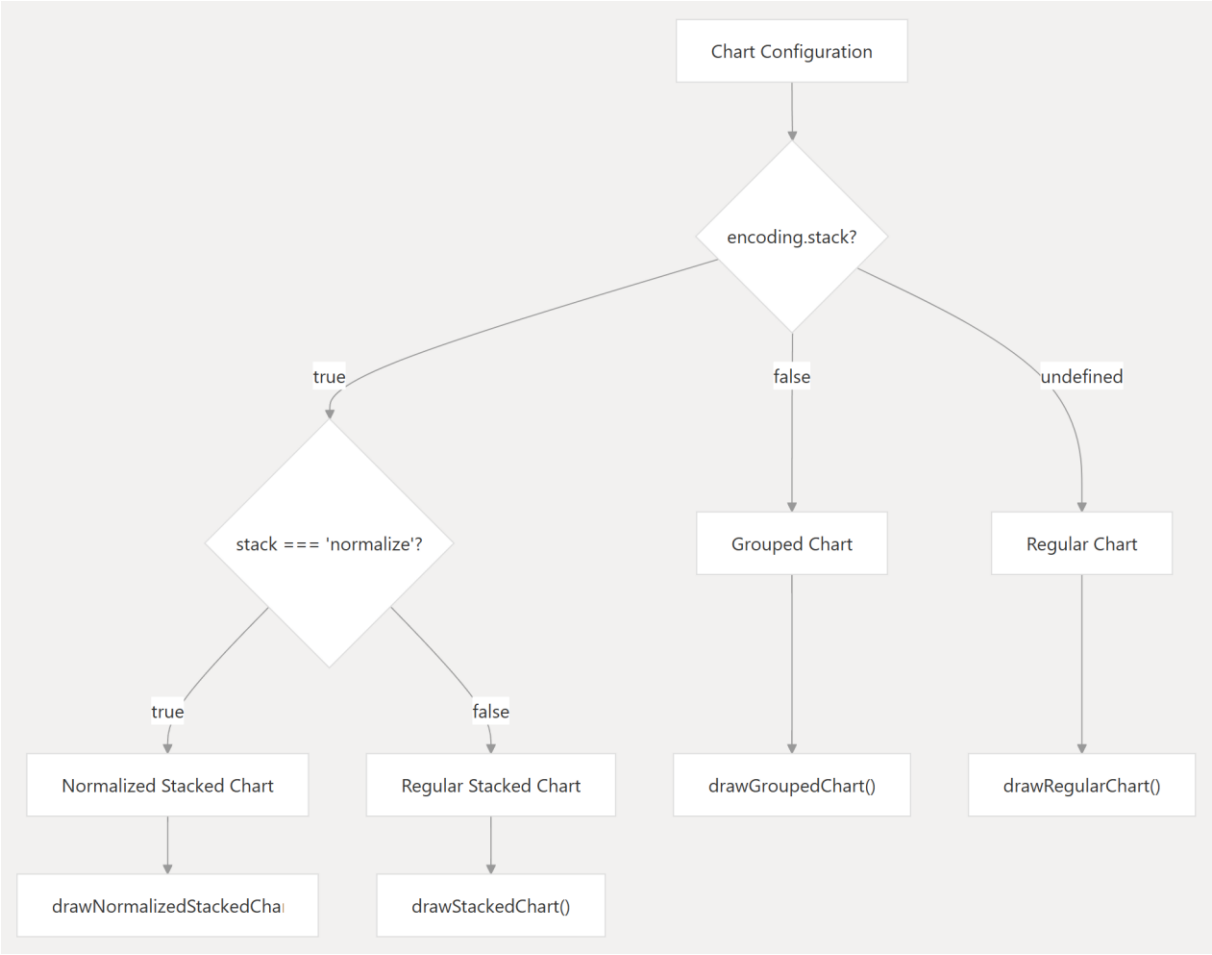

Figure 4.1 Block diagram for Bar Chart

- If “encoding.stack” is undefined, the system renders a Regular Bar Chart.
- If “encoding.stack” is set to true, a Stacked Bar Chart is created.
- If “encoding.stack” equals "normalize" (i.e., stack: "normalize"), a Percentage Stacked Bar Chart is rendered, where all values are normalized to 100%.
- If “encoding.stack” is false and the color parameter exists, the system renders a Grouped Bar Chart, in which each group represents a distinct category identified by the color value.

This decision logic is encapsulated in the following chart rendering flow:

Chart Configuration → Check “encoding.stack”

- Undefined → Regular
- True → Stacked
- “normalize” → Percentage Stacked
- False → Grouped

This logic ensures that the chart rendering engine dynamically selects and calls the appropriate rendering function depending on the encoding configuration.

The following sections detail the features and visualization results of each type.

4.1.1 Regular Bar Chart

The Regular Bar Chart is the most fundamental type of bar chart and is applicable when the input metadata includes two mandatory fields: "x" and "y". This chart is ideal for visualizing the one-to-one relationship between a categorical variable (x) and a quantitative value (y).

Key Features and Display Behavior:

- Required Fields: The chart requires "x" (categorical axis) and "y" (quantitative axis).
- Rendering Logic: Each unique value in the x axis corresponds to a single bar. The height (in vertical orientation) or width (in horizontal orientation) of each bar reflects the magnitude of the associated y value.
- Color Encoding: If the "color" field is specified in the encoding configuration, each bar can be colored according to its group or category. Users can also manually adjust these colors directly via the interactive legend.
- Orientation: The chart supports both vertical and horizontal orientations. This is configured using the direction parameter in the encoding object (e.g., direction: "horizontal" or direction: "vertical").
- Axis Label Rotation: To improve readability, users can customize the rotation angle of the axis labels using the axis.labelAngle property.
- Quantitative Scale Options: The chart supports various types of numeric scaling for the y axis. These options enhance flexibility when visualizing data with varying distributions or magnitude ranges. Including:
  - Linear Scale (ScaleLinear)
  - Power Scale (ScalePow)
  - Logarithmic Scale (ScaleLog)
- User Interactivity:
  - Tooltip: A dynamic tooltip is displayed when hovering over a bar, showing the corresponding x and y values.

– Interactive Legend: The legend not only displays color categories but also allows users to modify the color settings interactively.

Illustrations:

```
function drawVerticalChart1(data) {
    const barChart = document.querySelector('#verticalChart1');

    // Set configuration
    barChart.setAttribute("description", "Single Vertical - Population of countries");
    barChart.setAttribute("width", "500");
    barChart.setAttribute("height", "500");

    // Set data
    barChart.setAttribute("data", JSON.stringify({ "values": data }));

    // Set encoding
    barChart.setAttribute("encoding", JSON.stringify({
        "x": {
            "field": "name"
        },
        "y": {
            "field": "population",
            "axis": {
                    "labelAngle": 45,
                },
                    "scale": {
                        "type": "pow", //("type": "log")
                        "exponent": 0.5
                    }
        },
        "color": {
            "field": "randomLang",
            "scale": {
                "domain": [
                    "Belarusian language",
                    "English language",
                    "Romani language",
                    "Persian language"
                ],
                "range": "Reds"
            },
            "title": "Language"
        },
        "direction": "vertical"
    }));

    barChart.setAttribute("legend", true);
}
```

Figure 4.2 Metadata of Regular Bar Chart

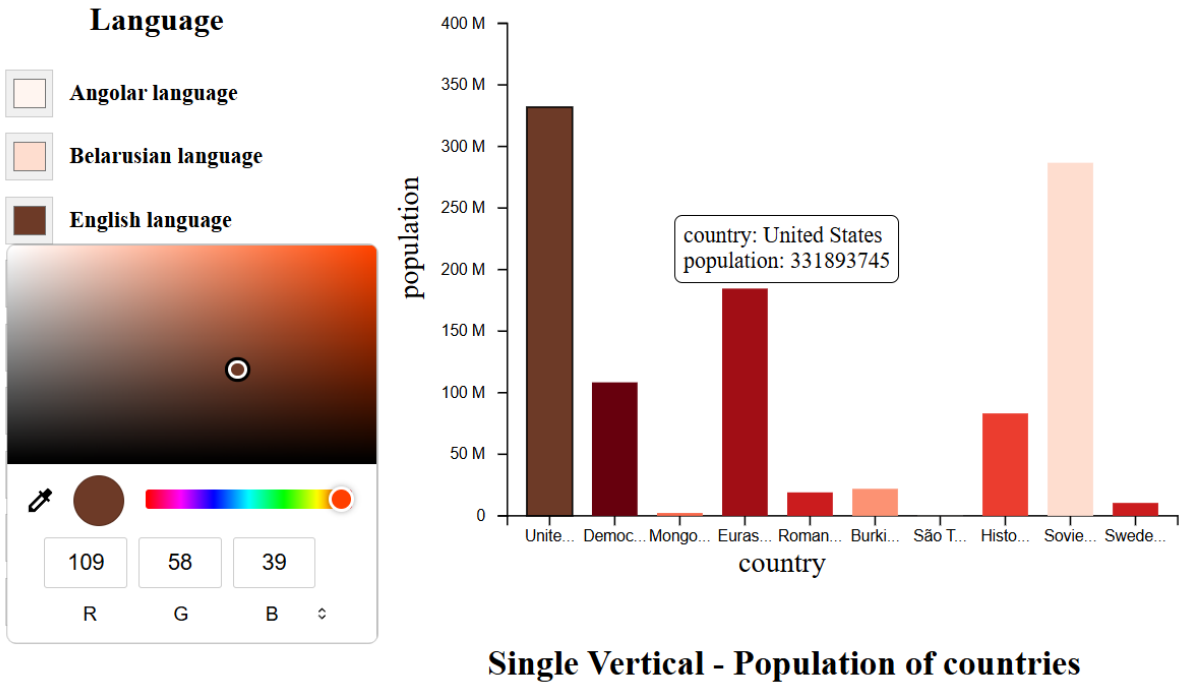

Figure 4.3 Visualization of Regular Bar Chart

4.1.2 Stacked Bar Chart

The Stacked Bar Chart provides a powerful way to visualize the composition of each category (x) by stacking sub-categories defined by the "color" field. This chart type is enabled when the "stack" parameter in the encoding is set to true.

Key Features and Display Behavior:

- Required Fields: The stacked bar chart requires three fields in the metadata:
  - "x": Categorical Axis (e.g., year, category label).
  - "y": Quantitative value for each sub-category.
  - "color": The field used to group and stack sub-categories within each x.
- Rendering Logic: For each unique value of x, a single bar is drawn. This bar is internally divided (or “stacked”) by the sub-categories specified in color, and each segment’s height (or width) corresponds to the associated y value.
- Stacking Calculation: The library uses d3.stack() to compute the start and end positions for each stacked segment, ensuring a smooth and accurate layout along the chosen axis.
- Orientation: Both vertical and horizontal bar orientations are supported through the direction parameter in the encoding object.

- Axis Label Rotation: To improve readability, users can customize the rotation angle of the axis labels using the axis.labelAngle property.
- Quantitative Scale Options: The chart supports the same quantitative axis scaling options as the regular bar chart:
  - Linear Scale (ScaleLinear)
  - Power Scale (ScalePow)
  - Logarithmic Scale (ScaleLog)
- User Interactivity:
  - Tooltip: When hovering over a segment, a tooltip appears showing the corresponding x, color, and y values, helping users understand the data distribution within each bar.
  - Interactive Legend: The legend provides a dynamic way to understand and control the color mapping. Users can also interactively modify colors directly from the legend to highlight or distinguish categories.

Use Case: This type of chart is especially useful when you want to analyze the total value across categories while still preserving the breakdown into contributing components (e.g., revenue by department per year).

Illustrations:

```
function drawStackedBarChart(data){
    const barChart = document.querySelector('#stackedBarChart');

    // Set configuration
    barChart.setAttribute("description", "Stacked Vertical - Number of films per year");
    barChart.setAttribute("width", "500");
    barChart.setAttribute("height", "400");

    // Set data
    barChart.setAttribute("data", JSON.stringify({ "values": data }));

    // Set encoding
    barChart.setAttribute("encoding", JSON.stringify({
        "x": { "field": "year" },
        "y": { "field": "count" },
        // "y": { "field": "count",
        //         "scale": {
        //             "type": "pow",
        //             "exponent": 0.5
        //         }},
        "color": {
            "field": "genre",
            "scale": {
                "domain": ["Children's television series", "Soap opera", "Drama", "Situation comedy",
"Music", "Anthology series", "Traditional pop", "Sitcom", "Adventure", "Historical drama", "Animation",
"Comedy"],
                // "range": d3.schemePaired
                "range": "Category10"
            },
            // "scale": [ "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd", "#e7ba52"],
            "title": "Movie genre"
        },
        "stack": true,
        "direction": "vertical"
    }));
    barChart.setAttribute("legend", true);
}
```

Figure 4.4 Metadata of Stacked Bar Chart

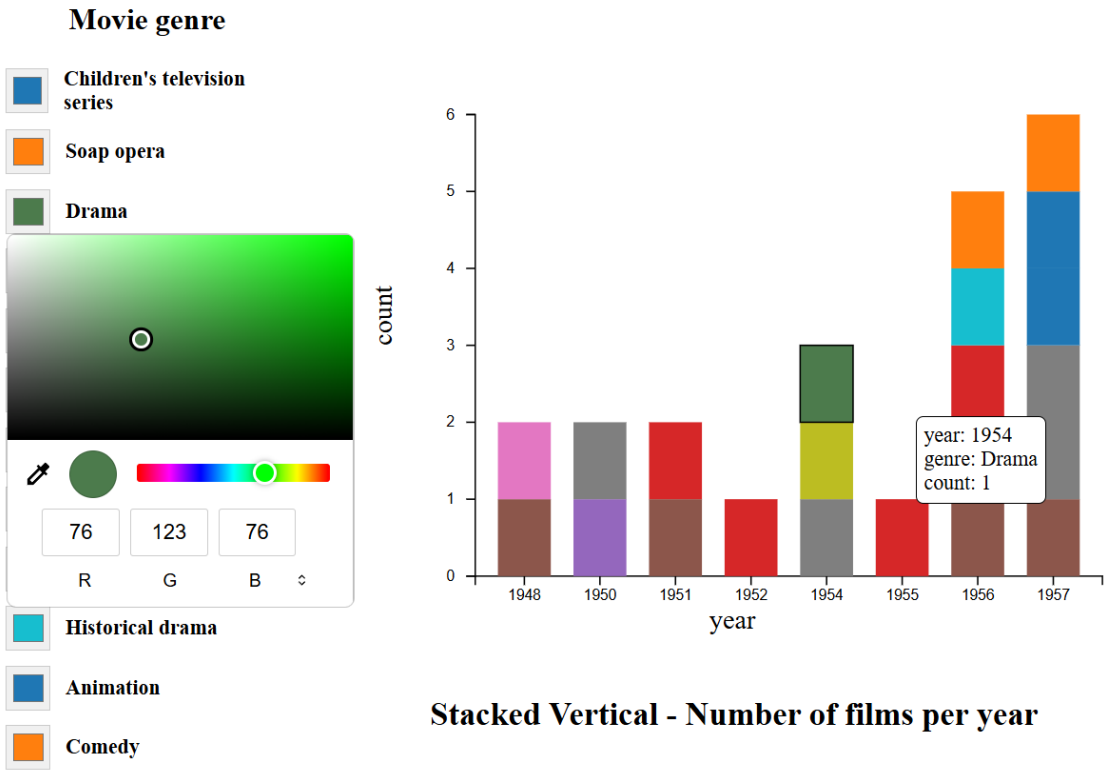

Figure 4.5 Visualization of Stacked Bar Chart

4.1.3 Percentage Stacked Bar Chart

The Percentage Stacked Bar Chart is a variation of the stacked bar chart that allows users to compare the proportional contributions of sub-categories across different main categories. This chart is triggered when the "stack" parameter in the encoding is explicitly set to "normalize".

Key Features and Display Behavior:

- Required Fields: This chart type requires three encoding fields that are similar to Stacked Bar Chart.
- Normalized Stacking: Instead of stacking segments according to absolute values, this chart type normalizes the total value of each bar (per x category) to 1 (or 100%). This transformation enables percentage-based comparisons between groups, regardless of their actual totals.
- Calculation Logic:
  - The stacking logic uses d3.stack() with normalization logic applied.
  - Each segment's size reflects its proportional contribution (y value divided by the total sum for that category).
- Orientation: Both vertical and horizontal bar orientations are supported through the direction parameter in the encoding object.
- Axis Label Rotation: To improve readability, users can customize the rotation angle of the axis labels using the axis.labelAngle property.
- Quantitative Axis Scaling:
  - While normalization ensures the values always sum to 100%, users can still specify the scaling method:
    - Linear (ScaleLinear)
    - Power (ScalePow)
    - Logarithmic (ScaleLog)
  - Note: In most cases, a linear scale is the most appropriate for percentage charts.
- User Interactivity:
  - Tooltip: Provides contextual information when hovering over a segment, including the corresponding x, color, and normalized y (percentage) value.

– Interactive Legend: Enables users to identify and recolor segments based on the color dimension, enhancing comparative analysis.

Use Case: This chart is especially effective for showing relative proportions within each category, such as displaying what percentage of total sales each product line contributes within different regions.

Illustrations:

```
function drawNormalizedStackedBarChart(data){
    const barChart = document.querySelector('#normalizedStackedBarChart');

    // Set configuration
    barChart.setAttribute("description", "Stacked Vertical - Number of films per year");
    barChart.setAttribute("width", "500");
    barChart.setAttribute("height", "400");

    // Set data
    barChart.setAttribute("data", JSON.stringify({ "values": data }));

    // Set encoding
    barChart.setAttribute("encoding", JSON.stringify({
        "x": { "field": "year" },
        "y": { "field": "count" },
        // "y": { "field": "count",
        //         "scale": {
        //             "type": "pow",
        //             "exponent": 0.5
        //         }},
        "color": {
            "field": "genre",
            "scale": {
                "domain": ["Children's television series", "Soap opera", "Drama", "Situation comedy",
"Music", "Anthology series", "Traditional pop", "Sitcom", "Adventure", "Historical drama", "Animation",
"Comedy"],
                // "range": d3.schemePaired
                "range": "Category10"
            },
            // "scale": [ "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd", "#e7ba52"],
            "title": "Movie genre"
        },
        "stack": "normalize",
        "direction": "vertical"
    }));
    barChart.setAttribute("legend", true);
}
```

Figure 4.6 Metadata of Percentage Stacked Bar Chart

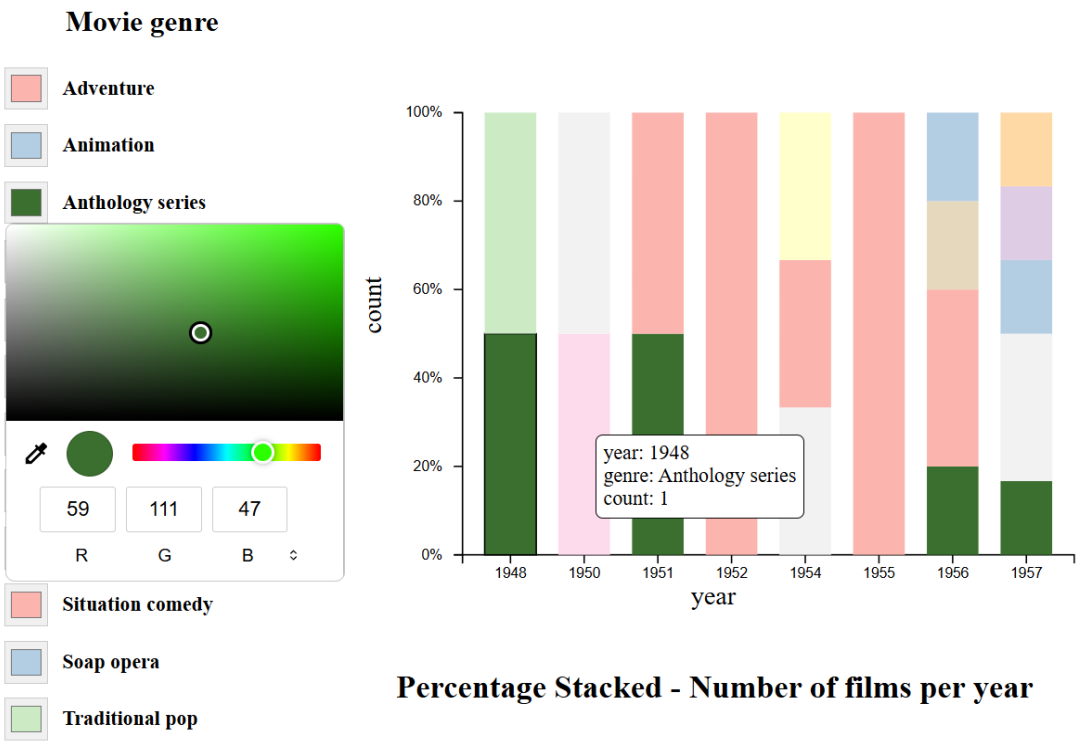

Figure 4.7 Visualization of Percentage Stacked Bar Chart

4.1.4 Grouped Bar Chart

The Grouped Bar Chart (also known as Clustered Bar Chart) is designed to compare multiple sub-categories across various categories. Instead of stacking segments atop one another, this chart places sub-bars side-by-side within each group, allowing for a clear visual comparison between sub-categories.

- Activation Conditions and Required Parameters:
  - The “color” parameter is present in the encoding.
  - The “stack” parameter is explicitly set to false (stack: false).
- Required encoding fields:
  - "x": The categorical variable that defines each group.
  - "y": The quantitative value of each bar.
  - "color": The sub-category used to split the bars within a group.
- Rendering Behavior and Data Layout:
  - For each unique value in the "x" dimension, the chart generates a group of adjacent bars, each representing a different "color" sub-category.
  - This layout provides a straightforward visual structure, making it easier to compare values across sub-categories within and between groups.

  - No stacking is involved; all bars start from a common baseline (typically the x-axis or y-axis origin, depending on orientation).
- Orientation and Axis Configuration:
  - The chart supports both vertical and horizontal orientations, controlled via the direction parameter in the encoding ("vertical" or "horizontal").
  - Users can customize the scale of the quantitative axis using:
    - Linear (ScaleLinear)
    - Power (ScalePow)
    - Logarithmic (ScaleLog)
- Axis Label Rotation: To improve readability, users can customize the rotation angle of the axis labels using the axis.labelAngle property.
- User Interaction and Customization Features:
  - Tooltip: Each bar provides a tooltip displaying the values of "x", "color", and "y" when hovered over.
  - Interactive Legend:
    - The legend reflects the values of the "color" field.
    - Users can interact with it to modify segment colors or filter visibility dynamically.
  - Label Angle and Styling: Axis labels can be rotated or styled using Vega-Lite's encoding properties for better readability.

Use Case: Grouped bar charts are ideal when the goal is to compare several categories side-by-side for multiple sub-groups, such as showing product sales across multiple regions or departments over time.

Illustrations:

```
function drawGroupedVerticalChart(data){
    const barChart = document.querySelector('#groupedVerticalChart');

    // Set configuration
    barChart.setAttribute("description", "Stacked Vertical - Number of films per year");
    barChart.setAttribute("width", "500");
    barChart.setAttribute("height", "400");

    // Set data
    barChart.setAttribute("data", JSON.stringify({ "values": data }));

    // Set encoding
    barChart.setAttribute("encoding", JSON.stringify({
        "x": { "field": "year" },
        "y": { "field": "count" },
        // "y": { "field": "count",
        //         "scale": {
        //             "type": "pow",
        //             "exponent": 0.5
        //         }},
        "color": {
            "field": "genre",
            "scale": {
                "domain": ["Children's television series", "Soap opera", "Drama", "Situation comedy",
"Music", "Anthology series", "Traditional pop", "Sitcom", "Adventure", "Historical drama", "Animation",
"Comedy"],
                // "range": d3.schemePaired
                "range": "Category10"
            },
            // "scale": [ "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd", "#e7ba52"],
            "title": "Movie genre"
        },
        "stack": false,
        "direction": "vertical"
    }));
    barChart.setAttribute("legend", true);
}
```

Figure 4.8 Metadata of Grouped Bar Chart

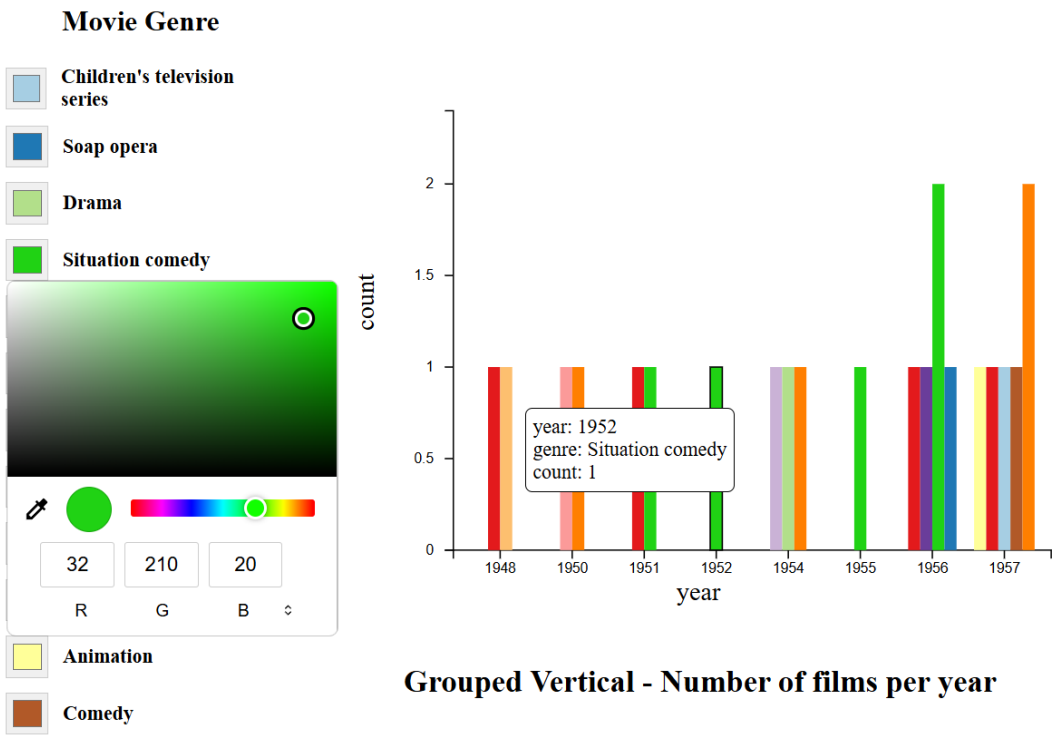

Figure 4.9 Visualization of Grouped Bar Chart

4.2 Pie Chart

The Pie Chart is a circular statistical graphic that is divided into slices to illustrate proportional data. It is commonly used to show percentage or proportional relationships among a small number of categories. Each slice of the pie represents a category and its relative value compared to the whole.

- Required Parameters:
  - "text": Acts as the label for each slice. This represents the name or category corresponding to each section of the pie chart.
  - "theta": Represents the quantitative value for each category, which determines the angular size (arc) of each slice. The larger the theta value, the bigger the corresponding slice.
  - "color": Used to assign different colors to each slice, enhancing distinction and readability between categories.
- Rendering and Behavior:
  - Each slice of the pie is proportionally calculated based on the total of all theta values.
  - The labels (text) are typically displayed either inside or outside the pie for clarity, depending on the chart configuration.
  - The use of the color parameter enables intuitive recognition of each section through distinct colors.
- User Interaction and Features:
  - Tooltip Support: When hovering over each slice, a tooltip is shown that displays the category name (text) and its corresponding value (theta), offering interactive feedback.
  - Legend Control:
    - A dynamic legend is generated based on the color field.
    - Users can interact with the legend to modify slice colors or filter categories on demand.
- Customization Options:
  - The chart can be further enhanced with options such as label placement, exploded views (offset slices), and percentage display, depending on Vega-Lite's extended configurations.

– Although typically static in structure, the pie chart supports dynamic updates and interactivity when embedded into web-based environments.

Use Case: Pie charts are most effective for displaying parts of a whole, such as market share, budget allocation, or survey results, where a small number of categories is involved.

Illustrations:

```
function drawPieChart(data) {
    const pieChart = document.querySelector('#pieChart');

    // Set configuration
    pieChart.setAttribute("description", "Pie chart - Countries with the largest population");
    pieChart.setAttribute("width", "400");
    pieChart.setAttribute("height", "400");

    pieChart.setAttribute("data", JSON.stringify({ "values": data }));

    // Set encoding
    pieChart.setAttribute("encoding", JSON.stringify({
        "text": {
            "field": "name"
        },
        "theta": {
            "field": "population"
        },
        "color": {
            "field": "randomLang",
            // "scale": {
            //     "domain": [
            //         "Belarusian language",
            //         "English language",
            //         "Romani language",
            //         "Persian language"
            //     ],
            //     "range": "Reds"
            // },
            // "title": "Language"
            // { radius: { "field": "population", "scale": { "type": "sqrt", "zero": true, "rangeMin": 20
} } }

        },
    }));
}
```

Figure 4.10 Metadata of Pie Chart

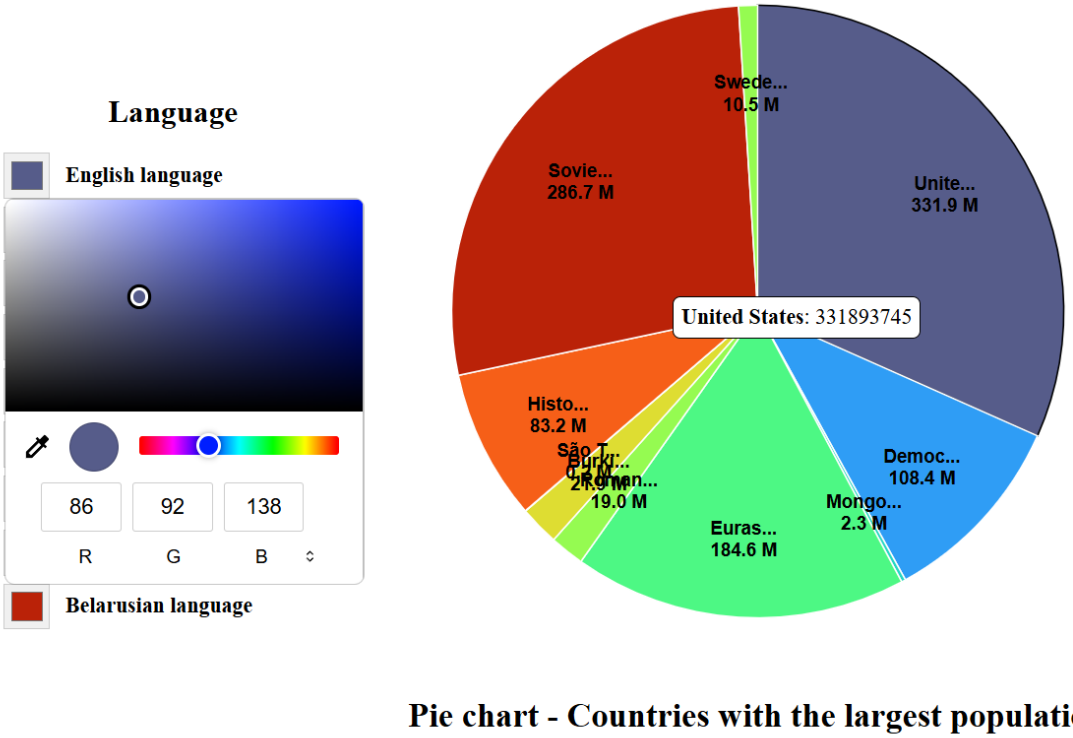

Figure 4.11 Visualization of Pie Chart

4.3 Geographic Map

The Geographic Map component in this project supports the visualization of spatial data using four types of maps: Choropleth Map, Connection Map, Bubble Map, and Hexbin Map. These visualizations enable users to represent geographic data in various forms, highlighting not only the spatial distribution of values but also connections between locations.

- Choropleth Map visualizes data by coloring geographic regions based on a particular metric.
- Connection Map displays a network of nodes and links (such as airports and flights) over a geographic layout.
- Bubble Map shows quantitative values as circles (bubbles) sized by magnitude and placed by coordinates.
- Hexbin Map groups data points into hexagonal bins, offering a density-based visualization of spatial patterns.

Each map type involves integrating a GeoJSON base map with attribute data, either through metadata bindings or, in the case of Bubble and Hexbin Maps, currently

through hardcoded inputs. Interactive features such as tooltips and legends are also included for intuitive exploration of spatial insights.

4.3.1 Choropleth Map

A Choropleth Map is used to represent statistical data through various shading or coloring of geographic regions. This type of map is particularly effective for displaying the distribution of a variable over predefined spatial areas, such as countries, states, or provinces.

- Required Parameters and Metadata Structure:
  - data file includes: id, data from geoJson (country name) and data from SPARQL query (population) corresponding to id.
  - “geoshape” mark to identify this as a choropleth map
  - Encoding fields such as id, label, and value from data.
  - projection: Specifies the type of geographic projection used to render the map (e.g., Mercator, Albers).
- Rendering Logic and Visualization Flow:
  - The data is connected to the geographic features using a lookup transform, allowing values to be joined with their corresponding regions in the GeoJSON.
  - Each region is colored based on the joined data attribute, typically a numerical variable (e.g., population, density, value index).
  - The visualization respects the selected projection, ensuring accurate spatial representation.
- Interactive Features:
  - Tooltip: When hovering over a region, a tooltip is displayed showing the identifier (id) of that region and the corresponding value retrieved via the lookup operation.
  - Legend: A dynamic legend is displayed to represent the color scale associated with the joined variable. It allows users to understand how colors map to values.

Use Case: Choropleth maps are ideal for visualizing socioeconomic, demographic, or environmental data across geographic regions where comparisons between areas are meaningful.

Figures:

```
function drawChoroplethMap(data) {
    document.querySelector('#choroplethMapData').setAttribute('data', JSON.stringify({
        "description": "Map Chart - Population of Countries",
        "width": 600,
        "height": 600,

        "data": {
            "values": data,
            "url": "https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/world.geojson",
        },
        "projection": {
            "type": "mercator"
        },
        "mark": "geoshape",
        "encoding": {
            "id": {
                "field": "isoCode"
            },
            "label": {
                "field": "countryLabel"
            },
            "value": {
                "field": "population"
            },
            "color": {
                "field": "population",
                "scale": {
                    // "range": d3.schemeOrRd[9]
                },
            },
            "shape": {
                "field": "geoShape",
                "type": "shape"  // hoặc "geojson"
            },
        }
    }));
}
```

Figure 4.12 Metadata of Choropleth Map

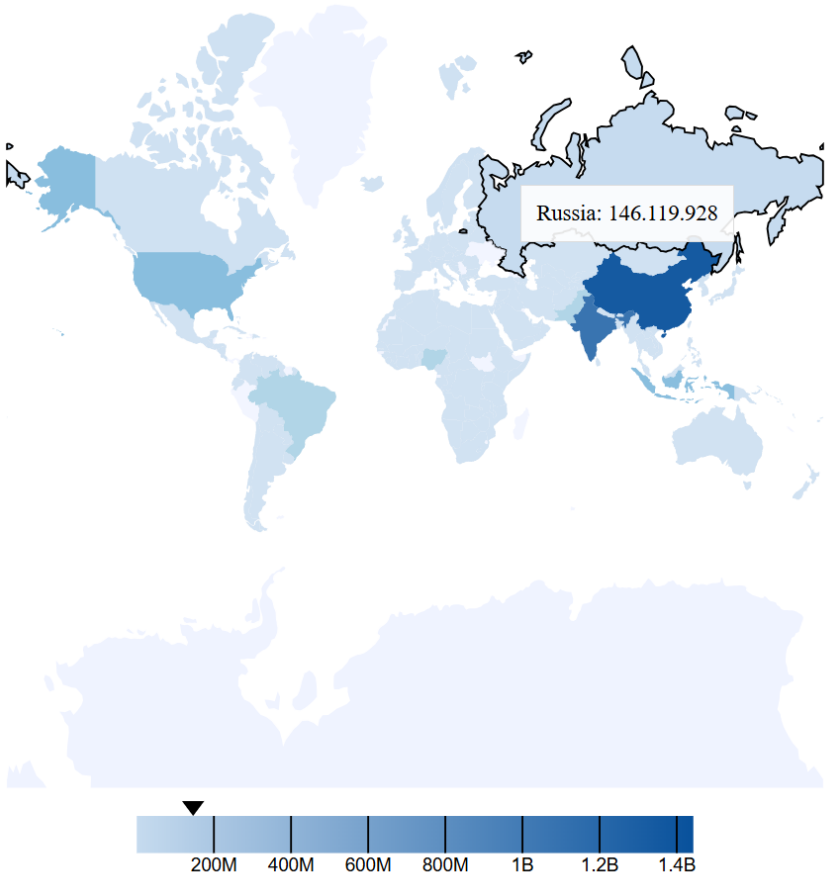

Figure 4.13 Visualization of Choropleth Map

4.3.2 Connection Map

The Connection Map is designed to visualize relationships between geographic nodes, such as transportation networks or communication paths, overlaid on a geographic base map. This type of map is particularly effective for illustrating how entities are interconnected across spatial locations—for example, showing flights between airports or data flows between cities.

- Architecture and Layered Structure: The Connection Map is built using a layered architecture, combining three distinct but overlapping layers:
  - Base Map Layer:
    - Contains the geographic layout (e.g., country or regional boundaries).
    - Utilizes a GeoJSON or TopoJSON file (such as us-10m.json) for rendering.
    - Defined via the "url" parameter in the data section.
  - Node Layer:

        - Displays key geographic entities (e.g., airports, cities) as points on the map.
        - Each node is positioned using longitude and latitude fields from a CSV or JSON file (e.g., airport.csv).
        - The node data file is connected to the map geometry via a lookup transformation, using a unique identifier.
    - Link Layer:
        - Renders the connections or relationships between nodes as lines or arcs (e.g., flight routes between airports).
        - Uses a separate dataset (e.g., flights-airport.csv) with source and target keys.
        - Includes geospatial coordinates (longitude, latitude) for each endpoint and matches nodes using a shared key field (e.g., "iata").
        - The "key" field in the "from" clause of the transform identifies how links are mapped to nodes.
- Key Parameters and Logic:
    - url (in each layer's data): Points to the file that contains the geographic or tabular data.
    - lookup (in transform): Specifies the column used to join between datasets, particularly for matching nodes to links.
    - key (in transform > from): Identifies the field in the linked dataset used to form the relationship.
- Interactive Features:
    - Tooltip: Hovering over a node or link reveals contextual information:
        - For nodes: the identifier (e.g., airport code or city name).
        - For links: details of the connection, such as source and destination pairs or frequency of interaction.
    - This interactivity enhances interpretability, allowing users to explore the structure and flow of the network.

Use Case: Connection Maps are ideal for illustrating networks such as airline routes, trade connections, migration flows, or any domain where spatially distributed entities are connected through meaningful relationships.

Figures:

```
function drawConnectionMap(data) {
    document.querySelector('#connectionMapData').setAttribute('data', JSON.stringify({

        "$schema": "https://vega.github.io/schema/vega-lite/v6.json",
        "width": 800,
        "height": 500,
        "layer": [
          {
            "data": {
              "url": "https://vega.github.io/vega-datasets/data/us-10m.json",
              "format": {
                "type": "topojson",
                "feature": "states"
              }
            },
            "mark": {
              "type": "geoshape"
            }
          },
          {
            "data": {
              "url": "https://vega.github.io/vega-datasets/data/airports.csv"
            },
            "mark": "circle",
            "encoding": {
              "longitude": {"field": "longitude", "type": "quantitative"},
              "latitude": {"field": "latitude", "type": "quantitative"}
            }
          },
          {
            "data": {
              "url": "https://vega.github.io/vega-datasets/data/flights-airport.csv"
            },
            "transform": [
              {"filter": {"field": "origin", "equal": "SEA"}},
              {
                "lookup": "origin",
                "from": {
                  "data": {"url": "https://vega.github.io/vega-datasets/data/airports.csv"},
                  "key": "iata",
                  "fields": ["latitude", "longitude"]
                },
                "as": ["origin_latitude", "origin_longitude"]
              },
              {
                "lookup": "destination",
                "from": {
                  "data": {"url": "https://vega.github.io/vega-datasets/data/airports.csv"},
                  "key": "iata",
                  "fields": ["latitude", "longitude"]
                },
                "as": ["dest_latitude", "dest_longitude"]
              }
            ],
            "mark": "rule",
            "encoding": {
              "longitude": {"field": "origin_longitude", "type": "quantitative"},
              "latitude": {"field": "origin_latitude", "type": "quantitative"},
              "longitude2": {"field": "dest_longitude"},
              "latitude2": {"field": "dest_latitude"}
            }
          }
        ]

    }));
}
```

Figure 4.14 Metadata of Connection Map

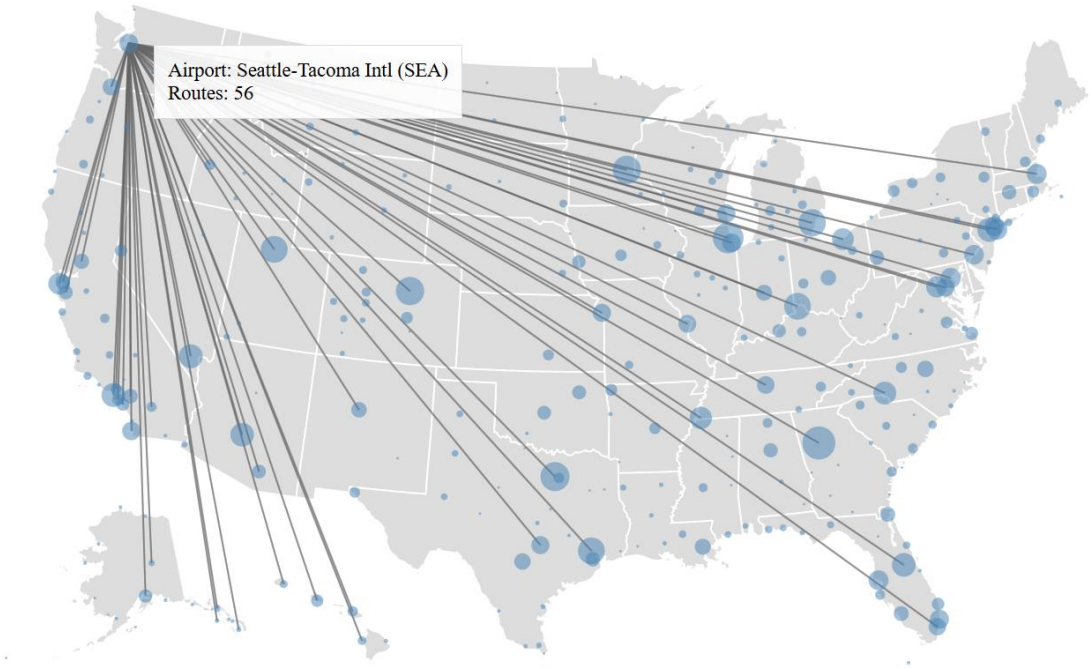

Figure 4.15 Visualization of Connection Map

4.3.3 Bubble Map

The Bubble Map is a form of geographic visualization where data values are represented by circles ("bubbles") plotted over specific coordinates on a map. The size of each bubble is proportional to a quantitative variable, making it effective for comparing magnitudes across different geographic locations.

Unlike other map types in the system, the current implementation of the Bubble Map does not yet support metadata-based automatic integration. Instead, the data is hard-coded within the configuration. The system distinguishes a chart as a Bubble Map using the flag: `"bubble": "true"`

- The Bubble Map is composed of two main data layers:
  - Base Map Layer:
    - Provides the geographic context (e.g., country or regional borders).
    - Implemented using a GeoJSON file, which is specified in the "url" parameter of the data field.
  - Bubble Layer:
    - Contains the dataset used to draw the bubbles.

        - This dataset is typically a CSV file that includes: longitude and latitude: Define the position of each bubble on the map. And n: Represents the magnitude or value that determines the bubble size.
        - The CSV file is manually defined within the visualization configuration.
- Key Visualization Logic:
  - The size of each bubble corresponds to the value of the "n" parameter. Larger values will produce larger circles, allowing users to quickly compare the intensity or quantity of data at different locations.
  - Coordinates from longitude and latitude ensure that each bubble is accurately positioned over the map's projected layout.
- Interactive Features:
  - Tooltip: When a user hovers over a bubble, a tooltip is displayed showing the value associated with that bubble (usually derived from the "n" field in the CSV).
  - This interactive feedback enables users to explore specific data points without cluttering the map with labels.
- Limitations:
  - The current implementation lacks support for dynamic metadata binding, meaning the chart must be configured manually rather than automatically adapting based on user-supplied metadata.
  - Future versions may improve usability by introducing support for metadata-driven generation similar to other chart types in the system.

Use Case: Bubble Maps are ideal for visualizing population sizes, event densities, or any geographic metric where a magnitude needs to be overlaid on a map in an easily comparable way.

Figures:

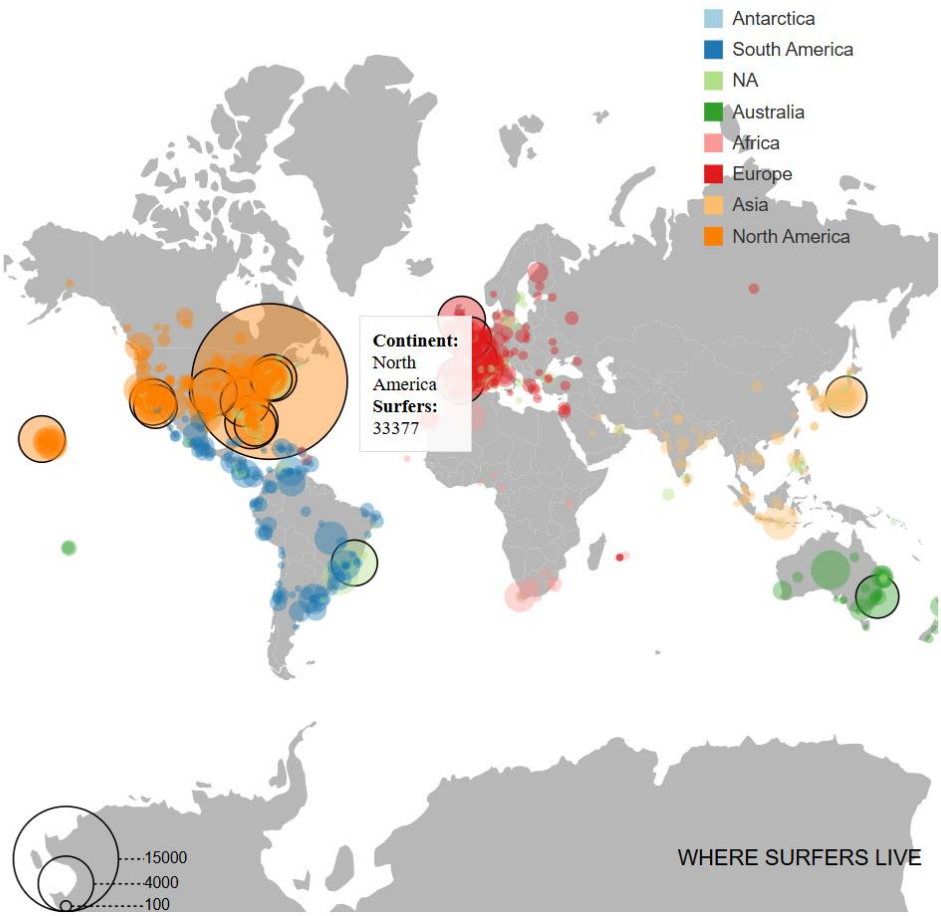

Figure 4.16 Visualization of Bubble Map

4.3.4 Hexbin Map

The Hexbin Map is a geographic data visualization technique used to display the density or frequency of points over a geographic area. Instead of individual bubbles or points, it groups spatial data into hexagonal bins, where each hexagon's color intensity corresponds to the number of data points within that area.

Structure and Data Sources: Similar to the Bubble Map, the Hexbin Map currently does not support metadata-based dynamic generation. The input data and configuration are hard-coded within the system. The visualization type is recognized using a specific flag: `"hexbin": "true"`

- The Hexbin Map consists of two primary components:
  - Base Map Layer:
    - Provides the background geography, such as countries or regions.
    - Implemented using a GeoJSON file, which is declared in the "url" field of the data configuration.

        - This file ensures the correct geographic layout on which hexagons will be rendered.
    - Hexbin Layer:
        - Represents the spatial distribution of data using hexagons.
        - Based on a CSV dataset that includes: longitude and latitude: Specify the coordinates of each data point. And n: A quantitative value that determines the density or magnitude used to color the hexbin.
        - This CSV file is manually integrated into the configuration and not dynamically bound via metadata at this stage.
- Hexbin Logic:
    - The map divides the geographical area into regular hexagonal grids.
    - Each grid cell (hexbin) counts the number of points (or aggregates values using "n") that fall within it.
    - The result is rendered as a series of hexagons, where the color of each hexagon is determined by the data magnitude.
- Interactive Features:
    - Tooltip: When the user hovers over a hexagon, the tooltip displays the value associated with that bin (e.g., count or aggregated "n").
    - Legend: A color legend is included to help users interpret the meaning of different color intensities, correlating color with magnitude.
- Limitations:
    - As with the Bubble Map, the Hexbin Map currently does not support metadata integration. All data and parameters are manually specified.
    - Further development could enable automated configuration via metadata to improve flexibility and usability.

Use Case: The Hexbin Map is especially effective when working with dense spatial datasets where individual points would overlap. Common use cases include: visualizing incident frequencies (e.g., crimes, tweets, sales) in high-density urban areas, traffic heatmaps, or environmental monitoring data.

Figures:

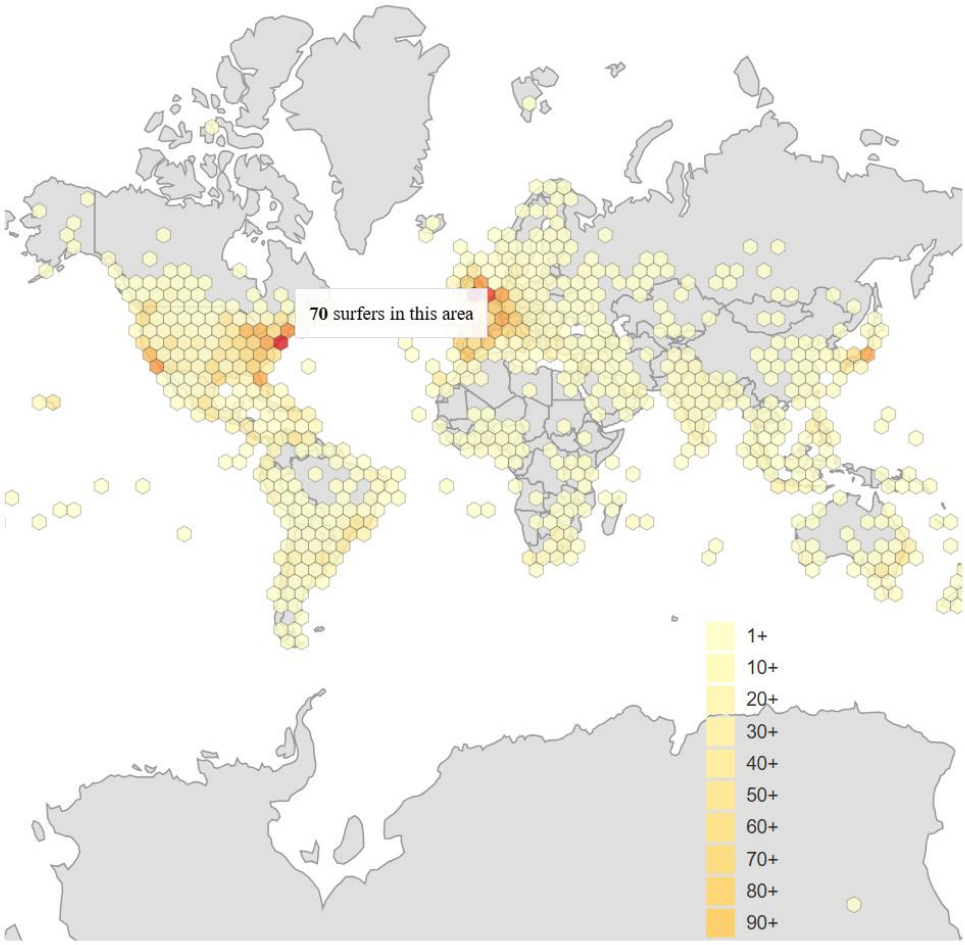

Figure 4.17 Visualization of Hexbin Map

4.4 Color Palettes

To provide flexibility and ease of use when customizing chart colors, the library includes a powerful color palette parsing function. This feature allows users to define palettes using intuitive string inputs like "Reds[5]", "Category10", or "Reds" — which are automatically mapped to their corresponding color schemes in the D3.js library, such as d3.schemeReds[5], d3.schemeCategory10, or d3.interpolateReds.

4.4.1 Purpose

- Enable users to define color palettes using a simple, human-readable syntax.
- Support both discrete palettes (e.g. d3.scheme*) and continuous palettes (e.g. d3.interpolate*).
- Normalize palette names automatically to ensure compatibility with D3's internal naming conventions.

4.4.2 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.

4.4.3 Syntax

The system supports a wide range of color schemes. Below are selected examples from each category to illustrate common use cases.

4.4.3.1 Categorical Palettes (Discrete)

Categorical palettes are ideal for distinguishing between distinct groups or categories, such as different regions or product types.

- "Category10" → Maps to d3.schemeCategory10, which returns an array of 10 distinct, high-contrast colors. (Use case: Assigning unique colors to different departments in a bar chart). Category10 (= d3.schemeCategory10)

- "Pastel1" → Maps to d3.schemePastel1, a softer palette suited for less aggressive visuals (Use case: Visualizing demographic segments in a pie chart). Pastel1 (= d3.schemePastel1)

4.4.3.2 Sequential Palettes (Continuous or Discrete)

Sequential palettes represent ordered data, such as rankings, values, or densities.

- "Reds[5]" → Maps to d3.schemeReds[5], returning five shades of red (Use case: Displaying population density by region in a choropleth map). Reds[k] (= d3.schemeReds[k]). *k in 3 – 11*.

- "Blues" → Maps to d3.interpolateBlues, offering a smooth gradient of blue tones (Use case: Indicating temperature changes or performance scores). Blues (= d3.interpolateBlues)

4.4.3.3 Diverging palettes

Diverging palettes are useful when the data has a meaningful midpoint, such as above/below average comparisons.

- "RdYlBu" → Maps to d3.interpolateRdYlBu, transitioning from red to yellow to blue (Use case: Highlighting deviations from a baseline, such as financial gains vs. losses). RdYlBu (= d3.interpolateRdYlBu)

- "Spectral[9]" → Maps to d3.schemeSpectral[9], offering a vivid diverging spectrum (Use case: Representing election results across a political spectrum). Spectral[k] (= d3.schemeSpectral[k]). *k in 3 – 11*.

4.4.3.4 Cyclical Palettes

Cyclical palettes are best suited for data with repeating or circular characteristics, such as angles or time-of-day. Rainbow (= d3.interpolateRainbow)

4.5 Color Function

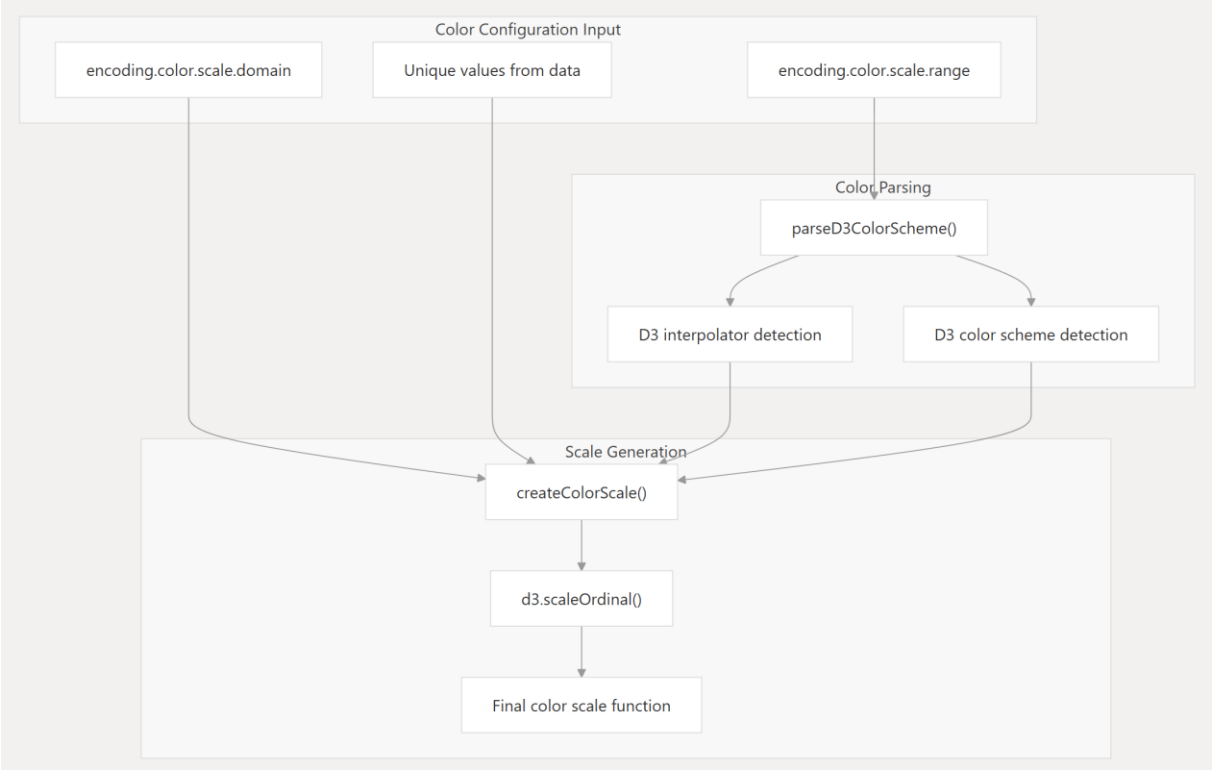

Figure 4.18 Block diagram for Color Function

4.5.1 Color Configuration Input

Three primary sources of input are considered:

- encoding.color.scale.domain:
  - A user-defined list of input values to be mapped to colors (e.g., ["A", "B", "C"]).
  - If not provided, it falls back to unique values extracted from the dataset.
- Unique values from data:
  - Automatically computed from the dataset if no explicit domain is given.
- encoding.color.scale.range:
  - A user-specified color scheme or interpolator name as a string (e.g., "Category10", "Reds[5]", or "interpolateViridis").

4.5.2 Color Parsing

The parseD3ColorScheme() function is responsible for interpreting the color range string:

- D3 interpolator detection:

  - Detects if the string refers to a continuous color interpolator (e.g., interpolateBlues, interpolatePlasma).
- D3 color scheme detection:
  - Checks if it refers to a discrete color scheme (e.g., schemeCategory10, schemeSet2, or schemeReds[5]).

4.5.3 Scale Generation

All parsed and derived inputs are passed to createColorScale() to construct a usable D3 color scale:

- It combines the domain (input values) and range (colors) to build the correct scale.
- For discrete (categorical) data, a D3 ordinal scale is created:
  `d3.scaleOrdinal(domain, range)`

4.5.4 Final Color Scale Function

The result is a color scale function, which maps input values to corresponding colors. This function is then used in the visual encoding process (e.g., assigning fill or stroke colors in a chart):

```
   const color = colorScale("A"); // Example output:
"#d73027"
```

4.6 Testing process

To ensure that the graph visualization library from SPARQL data works correctly and is easy to use as designed, it is necessary to build some test cases.

Name	Input	Test steps	Expected output	Result
Invalid SPARQL or Missing Fields	• SPARQL query with missing fields or typo	1. Provide data but the fields used for parameters x, y, ... do not exist. 2. Try rendering	• Error message or visual feedback.	PASS
Render Regular Bar Chart	• SPARQL query's result returning x and y fields	1. Call the web component "bar-chart" in HTML.	• Chart renders succesfully. • Labels appear correctly.	PASS

	• Metadata: x, y	2. Create the configuration for that component. 3. Use the SPARQL query's result for data. 4. Set the field for "x", "y". And set variable "true" for legend's attribute 5. Run visualization.	• Legend and tooltip are interactive.	
Render Choropleth Map	• Valid JSON metadata • SPARQL data returned and injected into "values" • A valid link to GeoJSON via "url"	1. Prepare SPARQL result with country isoCode, population, and countryLabel. 2. Inject data into the values field. 3. Load the map component. 4. Ensure the projection is set to mercator. 5. Render the map.	• Regions are displayed using geoShape. • Colors reflect population value by gradient. • Tooltip appears on hover showing countryLabel and population. • Legend shows color scale for population.	PASS
Toggle Legend Visibility	• Any chart metadata with "legend": false	1. Set chart type and provide metadata. 2. Set attribute "legend" is false in the configuration. 3. Render chart	• Legend hidden from chart.	PASS
Use the color palette provided by the library	• Valid JSON metadata • SPARQL data returned and injected into "values"	1. Set chart type and provide metadata. 2. In the encoding of configuration, enter value for "color" parameter.	• The "Reds" string will map to d3.interpolateReds in D3 library.	PASS

	• range field exists for color in encoding	3. In the color range field, enter a string whose name corresponds to the D3 color palette. Such as Reds 4. Render chart.	• The chart will display in interpolated red.	

Table 1. Test case table

Conclusion

This thesis presents the design and implementation of a lightweight, JavaScript-based visualization library tailored for SPARQL query results. With the growing adoption of semantic web technologies and the increasing volume of RDF data, the demand for accessible, interactive, and reusable tools to visualize SPARQL output is more critical than ever. However, existing visualization tools often lack support for SPARQL-native workflows or only provide limited chart types with minimal customization, posing a barrier to adoption—especially among users without advanced visualization expertise.

To address this gap, the project aims to develop a flexible and extensible library that allows users to create meaningful visualizations from SPARQL queries using simple metadata configuration. The library is built on top of the Vega-Lite visualization grammar, which provides a declarative approach to designing visualizations, offering both expressive power and ease of use.

The project is structured into several key chapters, each building up the theoretical foundation and practical implementation of the system:

- Chapter 1 reviewed existing literature in the fields of data visualization, RDF/SPARQL technologies, Vega/Vega-Lite grammars, and comparable tools. It highlighted the limitations of existing libraries when applied to semantic web data, setting the motivation for the proposed solution.
- Chapter 2 provided a detailed theoretical background, including formal definitions of RDF, SPARQL, metadata structure, and the Vega-Lite specification. It also established the foundation for how metadata can be used as an intermediary to map SPARQL results into visual representations.
- Chapter 3 outlined the system architecture, which follows a five-stage pipeline: SPARQL query input, metadata definition, visual encoding, rendering, and final output. A key design principle was to modularize each step so that future extensions (e.g., new chart types, new data formats) can be added with minimal refactoring. A clear block diagram illustrated how the

components interact, emphasizing which stages are handled internally by the library.

- Chapter 4 documented the implementation progress and current results. The library supports a range of bar charts (regular, stacked, grouped, and percentage-stacked), each customizable through metadata parameters like x, y, color, direction, and stack. A pie chart component was also implemented, using the theta, text, and color parameters. Furthermore, four types of geographic maps—choropleth, connection, bubble, and hexbin—were developed, leveraging GeoJSON and CSV files to bind data to spatial elements. Each component supports interactive features such as tooltips, legends, and dynamic encoding options. Additionally, all components share a common configuration pattern, supporting attributes like width, height, description, and legend, enhancing usability and consistency.

Final Reflections

The proposed visualization library represents not only a technical solution but also a conceptual bridge between semantic data and intuitive visual analytics. By abstracting away the complexities of SPARQL syntax and Vega-Lite grammar, it lowers the barrier for RDF data consumers to explore and communicate insights effectively. The metadata-driven approach makes the system extensible and adaptable, paving the way for further community contributions and domain-specific customizations.

Moreover, the project reinforces the importance of visualization in the semantic web ecosystem—not just as a tool for presentation, but as an integral part of exploration, validation, and storytelling. The practical implementation demonstrates that RDF data, often seen as abstract and difficult to digest, can be made visually compelling and user-friendly through thoughtful system design.

In conclusion, this work delivers a foundation for a reusable, semantic-aware visualization framework and contributes meaningfully to the intersection of linked data and visual analytics. While much has been accomplished, the potential for future growth remains substantial, and this project lays the groundwork for that continuing evolution.

Future Development Directions

While the current version of the SPARQL visualization library has achieved several core functionalities across various chart types, there are still many promising directions for further development to improve scalability, usability, and extensibility. These future improvements are outlined below:

Expansion of Visualization Types

The library currently supports fundamental chart types including bar charts, pie charts, and geographic maps. In the future, the following chart types could be added to support more complex analytical needs:

- Line Charts and Area Charts for temporal or trend-based data.
- Tree Maps and Sunburst Diagrams for hierarchical data visualization.
- Network Graphs for representing RDF structures and SPARQL result sets with node-link relationships.
- Scatter Plots with Regression Lines for analytical insight in numerical data sets.

Enhanced Metadata Inference

The current system requires users to explicitly define metadata for encoding. Future versions can integrate:

- Automatic metadata inference from SPARQL result structures.
- Schema-aware visualization suggestions based on RDF vocabularies (e.g., rdfs:label, rdf:type, owl:sameAs) and statistical analysis of result types.
- This would allow less-experienced users to receive chart suggestions automatically without writing detailed metadata.

Visual Configuration Interface

To enhance usability for non-programmers:

- Develop a graphical user interface (GUI) for generating chart metadata through form-based inputs or drag-and-drop interfaces.
- Offer real-time preview updates as users modify parameters such as axis mapping, colors, and chart type.

Integration with External Knowledge Sources

Extend support for federated queries and automatic enrichment using:

- Linked Open Data endpoints like DBpedia, Wikidata, or GeoNames.
- Schema.org or domain ontologies to suggest relevant fields for visualization or grouping. Offer real-time preview updates

Improved Interactivity and User Experience

While the current charts support basic interaction (tooltips, zoom, highlight), future versions may provide:

- Linked visualizations that allow brushing and filtering across multiple charts.
- Custom interaction layers, such as filtering with sliders or search bars integrated with SPARQL queries.

Documentation and Developer Community

To increase adoption and collaboration:

- Provide comprehensive documentation, examples, and tutorials.
- Launch an open-source repository and build a community around contributions and issue tracking.

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/.