String Charter 3: A Web App for Visual Transport Schedules

Martin Rabensteiner and Stephan Robinig

DAT.C406UF Information Visualisation 3VU SS 2025
Graz University of Technology

30 Jun 2025

Abstract

This report describes String Charter 3, an open-source web application for visualising transport schedules as string charts. Schedule data can be loaded in GTFS format. Many transport providers make their timetables available in this format, although sometimes some massaging and cleaning of the dataset is necessary.

String Charter 3 builds on the previous String Charter 2, which had more limited functionality and a more minimal user interface. The Tauri framework for cross-platform desktop deployment was updated from Tauri 1 to Tauri 2, NPM was replaced by Yarn for package management, and Bootstrap was replaced by Tailwind for responsive UI styling. Key improvements include a redesigned user interface with structured side and top menus, enabling streamlined uploading and selection of GTFS data, interactive route filtering, and customisable visual settings. These enhancements significantly improved usability and accessibility, facilitating efficient exploration and comparison of public transport schedules. A live demo of the application is available on GitHub Pages

Contents

List of Figures

Chapter 1

Introduction

This report describes the development String Charter 3, an open-source web application for visualising transportation schedules in GTFS format as string charts [Robinig et al. 2025b]. A live demo of the application is available on GitHub Pages [Robinig et al. 2025a].

1.1 String Charts

String charts are a way of visualising transportation data on a two-dimensional timeline, usually with time on the x-axis and the route (stops) on the y-axis. They are not only used by passengers, but also in planning and analysing timetables. They can be useful to improve routes, dispose resources, such as trains, and to plan encounters on single-track routes. A related survey of string charts and string charting tools is available [Nepal et al. 2025].

1.2 String Charter 2

String Charter 2 was a project by Inge Gsellmann, Michael Hebesberger, and Danijela Lazarevic in 2023 [Gsellmann et al. 2023]. The aim was to build a web application to visualise publicly accessible transportation schedules in GTFS format as string charts. The web app realises this in a simplistic approach with no options to adapt the visualisation. The stations are not sanitised, potentially leading to misinterpreted routes. For example, in Figure 1.1, routes towards Graz are cut before Bruck an der Mur, because the stations there are distinguished between platforms.

The authors agreed to place their code under an MIT licence, which made it possible to develop String Charter 3 based on the existing code. Although most of the user interface and the graph generation were rewritten, the parsing of the GTFS data was a convenient base to build to new visualisation on and remained almost unchanged.

1.3 Other Related Work

Besides String Charter 2, other tools also inspired the development of String Charter 3.

In String Charter 2, stations are equally spaced in the string chart, regradless of the physical distance between them. Other tools, for example jTrainGraph [Scherzinger 2022], represent the distance between stations according to the actual distance between them. This gives a better sense of how far stations are away from each other and how fast trains travel.

MBTA Viz visualises the data provided by the Massachusetts Bay Transit Authority (MBTA) in Boston, USA [Barry and Card 2014]. The developers Mike Barry and Brian Card show several possibilities to visualise the given data. One of those is the comparison of divergent travelling times. During peak hours, especially in the morning and the afternoon, trips take longer because of higher traffic and increased passenger numbers. This was taken as inspiration for a corresponding new feature in String Charter 3.

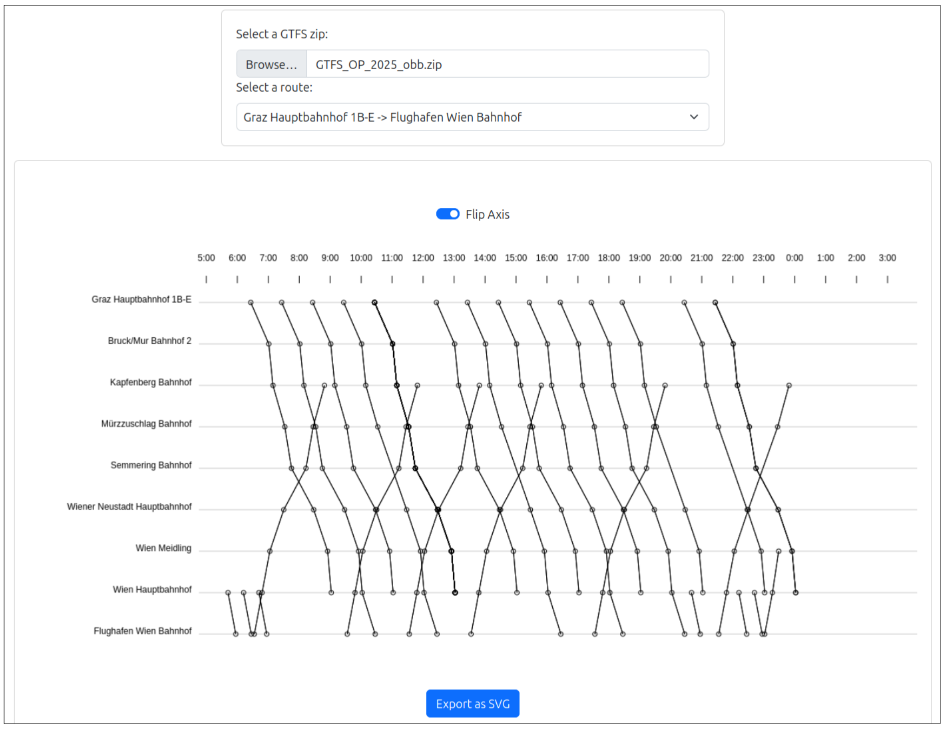

Figure 1.1: String Charter 2 showing a route from Graz to Vienna.

Chapter 2

Tech Stack and Deployment

One main focus of String Charter 3 was to modernise and streamline the existing tech stack. This includes the adoption of Yarn for more efficient package management, transitioning to the Vue framework for reactive and component-based frontend development, and utilising Gulp to automate and optimise the build processes. Additionally, the integration of Tailwind CSS allows for rapid UI development with utility-first design, while deploying via GitHub Pages ensures a lightweight and version-controlled hosting solution. These tools collectively contribute to improved maintainability and enhanced user experience.

2.1 Yarn

Yarn is a fast, reliable, and secure package manager for JavaScript projects [Yarn 2025], developed to address some limitations found in NPM. It offers deterministic dependency resolution through a lock file (`yarn.lock`), ensuring consistent installs across different environments. Yarn also improves performance by parallelising package downloads and caching them locally, which speeds up subsequent installs. Additionally, Yarn provides better offline support and enhanced security checks. These features make Yarn a preferred choice over NPM in many projects, especially those requiring reproducible builds and faster dependency management.

2.2 Tauri

Tauri is an open-source framework for building lightweight, secure, cross-platform desktop applications using web technologies such as HTML, CSS, and JavaScript [Tauri 2025a]. Unlike traditional Electron-based apps, Tauri produces significantly smaller binaries by leveraging the system's native web renderer and a Rust-based backend, enhancing performance and security. It supports integration with modern frontend frameworks and offers built-in features like code signing, auto-updates, and a flexible API to access native OS functionality.

2.3 Vue

Vue.js is a progressive JavaScript framework for building user interfaces and single-page applications [Vue.js 2025]. It is designed to be incrementally adoptable, allowing developers to integrate it into existing projects easily. Vue emphasises a component-based architecture, reactive data binding, and a simple, flexible API that facilitates rapid development and maintainability. Its lightweight size and strong ecosystem make it a popular choice for modern web development.

2.4 Gulp

Gulp is a streaming build system and task runner for automating repetitive development tasks such as minification, compilation, and testing [Gulp 2025]. It uses a code-over-configuration approach, allowing developers to write tasks in JavaScript that can process files through a pipeline, improving build efficiency and maintainability. Gulp's use of streams enables fast file transformations and reduces overhead compared to traditional build tools. Gulp is used in the project to provide a `clean` and `cleanAll` utility, automating the removal of build artefacts and temporary files to ensure a fresh build environment.

2.5 Tailwind

Tailwind is a utility-first CSS framework designed to enable rapid UI development by providing low-level, reusable CSS classes directly in HTML [Tailwind 2025]. Utility-first means that instead of writing custom styles, developers compose designs using classes for margin, padding, colour, typography, and so on. This approach encourages consistency, reduces stylesheet bloat, and improves maintainability. Tailwind also offers extensive customisation options and integrates well with modern JavaScript frameworks.

CSS classes exist for several CSS properties that can be appended to HTML elements. No self-written classes or CSS stylesheets are needed. The benefits of using Tailwind are:

- *Faster development*: Compose UI directly in markup without switching between CSS and HTML files.
- *Highly customizable*: Configurable design tokens (colours, spacing, fonts).
- *Consistent styling*: Utility classes encourage uniform design patterns.
- *Small final CSS size*: Purging unused styles leads to minimal CSS.
- *Responsive and state variants*: Easily add breakpoints and pseudo-classes.

2.6 GitHub Pages

GitHub Pages is a free hosting service provided by GitHub for publishing static websites directly from a GitHub repository. It allows developers to deploy web applications, documentation, or project showcases without the need for external hosting infrastructure. GitHub Pages supports HTML, CSS, JavaScript, and static site generators, and integrates seamlessly with version control workflows. This makes it an ideal platform for sharing demos, visualisations, or research tools, such as the live version of the String Charter 3 application.

2.7 GitHub Pages Deployment Process

Deployment to GitHub Pages is handled through the `gh-pages` package and a dedicated Yarn script.

- In `package.json` the repository is declared as:

  ```
  "repository": "git@github.com:StofflR/String-Charter-3.git"
  ```

  ensuring that `gh-pages` can detect the correct remote.

- A new script entry:

  ```
  "deploy": "gh-pages -d dist"
  ```

  publishes the contents of the compiled output folder (`dist/`) to the `gh-pages` branch. The `-d dist` flag tells `gh-pages` where the production build is located.

- The deploy command must be executed inside a working Git clone with the necessary push permissions; otherwise, the publication step will fail.

Since the project imports non-standard asset types (SVG, ICO, ZIP), custom TypeScript declaration files are added:

```
declare module '*.svg' {
  const content: React.FunctionComponent<React.SVGAttributes<
    SVGElement>>;
  export default content;
}
declare module '*.ico' { const content: string; export default
    content; }
declare module '*.zip' { const content: string; export default
    content; }
```

These declarations are placed in file `custom.d.ts`, and `tsconfig.json` is extended to include them:

```
"include": [
  "src",
  "src/**/*",
  "custom.d.ts"
]
```

After running the usual build step (`yarn build`), invoking `yarn deploy` pushes the freshly built site to the `gh-pages` branch, making the application available at `<user>.github.io/String-Charter-3`.

2.8 Vite Configuration

To ensure that the application is served correctly from the `gh-pages` branch (where the site will not be hosted at the domain root but at `/String-Charter-3/`), two parameters must be set properly in `vite.config.ts`:

- `root: './'` — sets the project root so that Vite resolves entry points relative to the repository's top-level folder.
- `base: './'` — forces Vite to emit all asset URLs as *relative* paths. Without this setting the generated HTML would reference absolute paths (e.g. `/assets/...`), which break when the site is served from a sub-directory on GitHub Pages.

With these options, the build in `dist/` can be published directly by the `gh-pages -d dist` script, and all links to scripts, styles, and images resolve correctly in the deployed environment.

Chapter 3

User Interface

The second key area of improvement in String Charter 3 was the user interface. The goal was to move away from a traditional web page appearance and toward a more application-like look and feel. A primary focus was placed on the ease of controls, ensuring that interactions are intuitive and efficient for the user. By leveraging modern frontend technologies and a more structured design system, the UI now offers a cleaner, more responsive, and user-friendly experience. The user interface of String Charter 3 is shown in Figure 3.1.

3.1 Application

The application uses a singleton instance to manage and pass references across different parts of the codebase, rather than relying on prop drilling through the Vue component hierarchy. This approach simplifies the architecture by centralising shared state and functionality, making it easier to access key objects (such as the chart generator or configuration settings) from anywhere in the application. The structure of the singleton is shown in Figure 3.2. While this deviates from Vue's conventional reactive data flow, it is well-suited for this context where interactivity is relatively contained and a lightweight structure is preferred.

3.2 Menu Bar

The application's menu bar provides essential global controls for managing data and navigating between different features. It includes options to upload and select GTFS data files. Additionally, it allows users to toggle between the different views of the side menu, such as data selection, route filtering, and visual configuration. The menu bar also offers the ability to export the generated chart as an SVG file. This export functionality currently relies on an experimental file save dialogue feature [MDN 2025], which is not yet fully supported across all browsers and may require further refinement. To ensure usability, a fallback mechanism is implemented that defaults to the standard browser download behaviour, saving the file to the user's designated download folder. Lastly, the menu bar includes access to an About dialogue, which displays information about the application's purpose and version.

3.3 Top Bar

The top bar of the application provides a set of high-level controls that affect the overall layout and interpretation of the generated chart. Users can flip the axis of the chart to switch between horizontal and vertical orientation, depending on their visualisation preference or the format of the data. Figure3.3 shows a string chart generated by String Charter 3. A comparison mode can be enabled, which allows differences between route datasets (such as delays or schedule deviations) to be visualised directly within the chart. An example can be seen in Figure3.4. Additionally, a quick-select feature is available, enabling

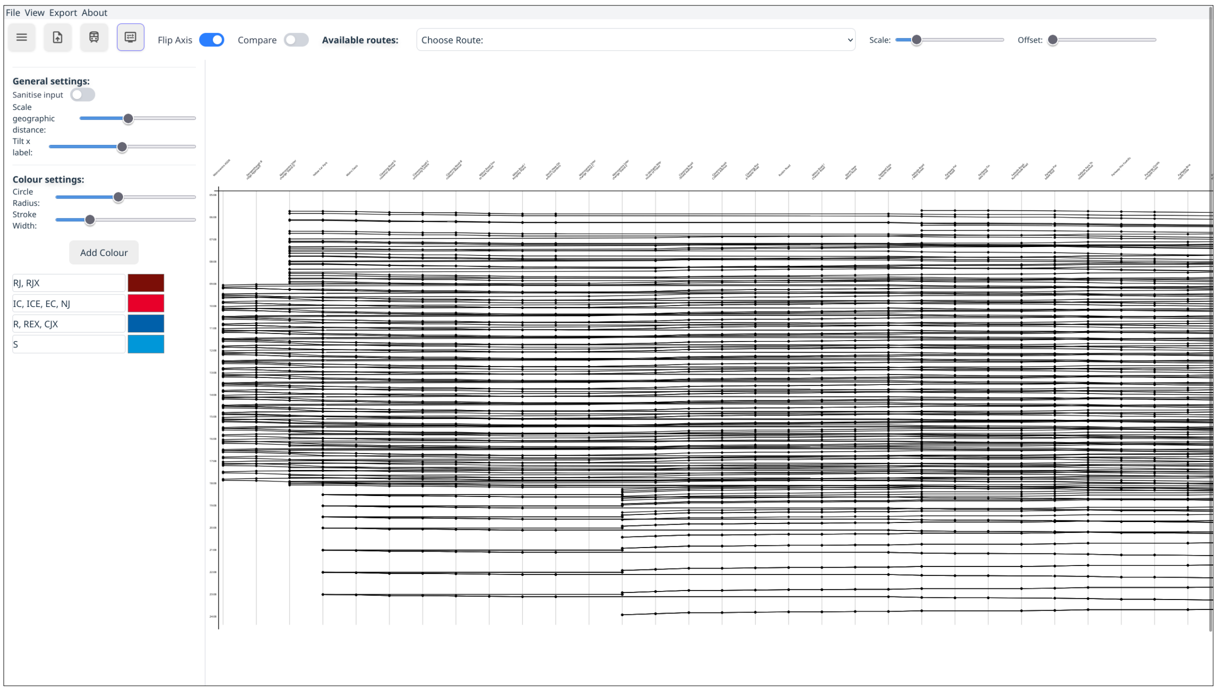

Figure 3.1: String Charter 3: User interface.

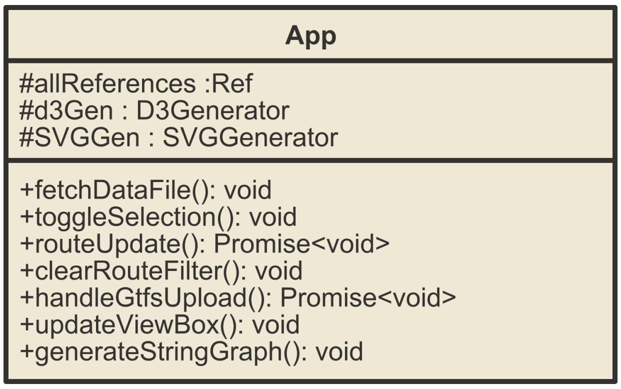

Figure 3.2: String Charter 3: A singelton app instance centralises shared state across the application.

users to quickly select specific routes from the dataset. Further controls allow users to apply scaling and offset adjustments to the entire plot, providing greater flexibility in aligning and resizing the visualisation to fit different display contexts or aesthetic requirements.

3.4 Side Bar

The application's left side bar is divided into three main panels, each supporting a distinct part of the chart generation workflow. The first panel allows users to select and upload GTFS data in ZIP format. By default, two datasets are preloaded: one from ÖBB (Austrian Federal Railways) and one from Go Northeast, enabling users to experiment immediately without requiring manual input. The second panel provides tools for filtering and selecting specific routes from the dataset. Users can select multiple routes for visualisation; however, this is only meaningful when the selected routes are structurally similar, such as variants of the same line. The third panel contains visual configuration settings, including options for input sanitisation, adjusting geographic scale between stations, styles for stop circles and route lines (e.g., width, dashing), and defining colouring rules based on string matching. These controls allow for

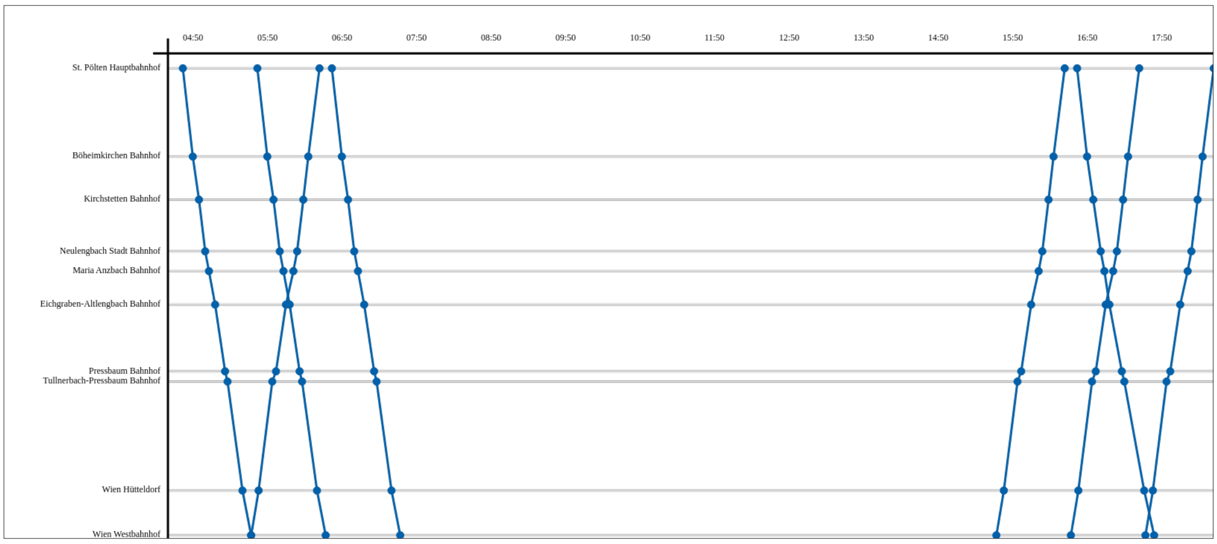

Figure 3.3: String Charter 3: Generated string chart.

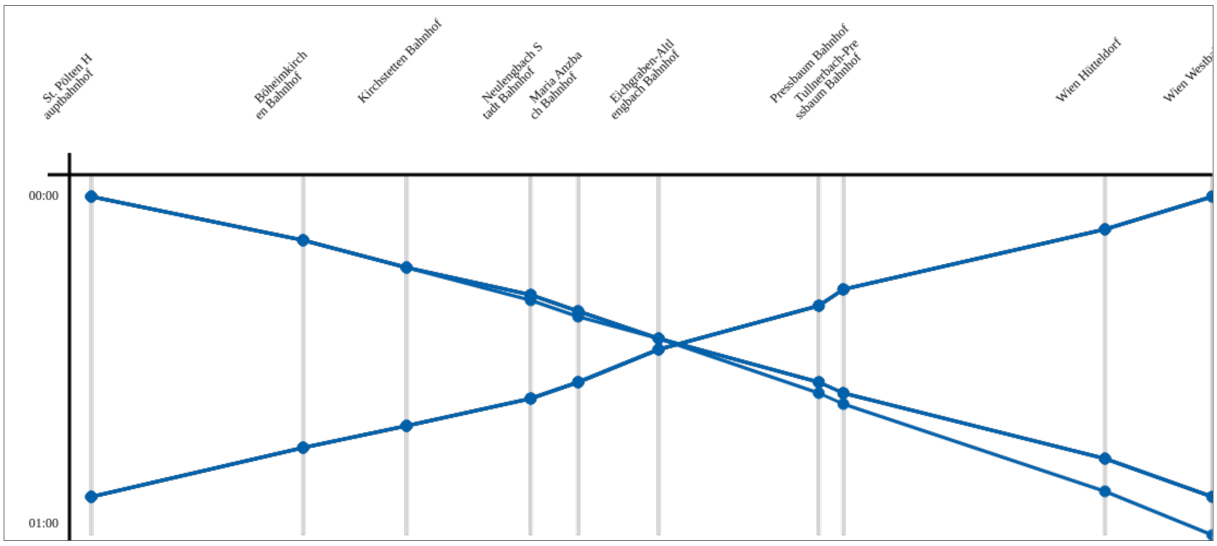

Figure 3.4: String Charter 3: Generated comparison chart.

fine-tuned customisation of the generated chart output.

Chapter 4

Chart Generation

The chart generation logic has been modularised and virtualised in String Charter 3 to improve flexibility and reusability. The main objective was to introduce a template class that serves as a common interface for various chart drawing implementations. This abstraction simplifies the integration of different plotting libraries by providing a consistent structure, reducing duplication, and enabling easier maintenance and extension.

4.1 Base Class

The `StringChartGenerator` class is an abstract base class designed to facilitate the creation of charts represented as `string` outputs. It provides a foundational interface and utility methods for derived classes that generate chart data in formats such as HTML, SVG, or D3, by abstracting away the specific implementation details of different rendering techniques. This allows users to generate charts without needing to know how they are internally constructed or what rendering technology (e.g., canvas, D3) is used. Subclasses implement the chart generation logic while adhering to a consistent interface and visuals.

Any subclass of `StringChartGenerator` must implement a set of abstract methods that define the core drawing primitives used for chart construction. These include methods for drawing lines, rendering text, placing circular markers, and adding diagonal text elements. Specifically, the methods `drawLine`, `drawText`, `drawStopCircle`, and `drawDiagonalText` must be implemented to provide concrete rendering behaviour appropriate to the target output format. This ensures that all derived classes offer a consistent interface for chart generation while allowing flexibility in how the graphical elements are encoded or displayed.

Classes such as `CanvasGenerator`, `SVGGenerator`, or `D3Generator` extend this abstract class and implement the `generate()` method to output string charts in the specific formats. This is illustrated in Figure 4.1.

4.2 D3

D3.js is a powerful JavaScript library for producing dynamic, interactive data visualisations using web standards such as SVG, HTML, and CSS [Bostock 2025]. It provides low-level access to the document object model (DOM) and supports binding data to visual elements, enabling highly customisable charting and data interaction capabilities.

One concrete implementation of the `StringChartGenerator` base class is tailored specifically for D3.js. This subclass translates abstract drawing instructions into D3-compatible SVG markup, allowing the chart to be embedded or rendered as part of a web page using standard D3 techniques.

D3.js was chosen as the rendering backend for one implementation of the `StringChartGenerator` class due to its powerful and flexible support for generating SVG elements programmatically. It allows for

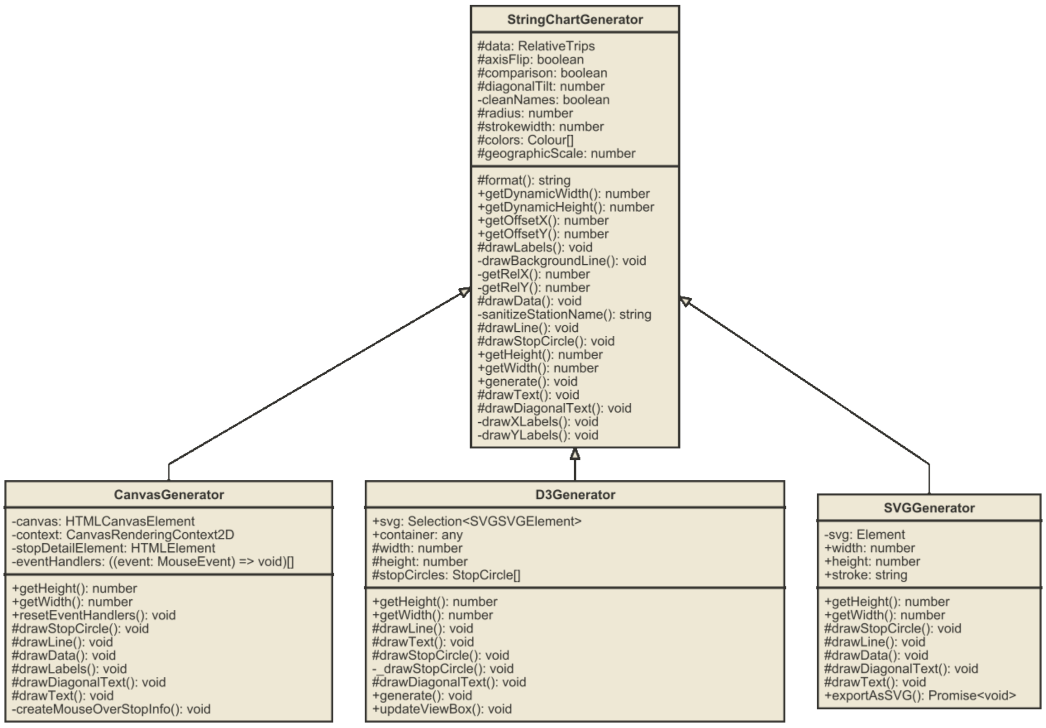

Figure 4.1: String Charter 3: Chart generation class structure.

precise control over graphical primitives and makes it straightforward to construct complex visualisations directly from data. Additionally, D3 offers built-in mechanisms for handling user interactions such as mouse hover events, which are essential for enhancing the interactivity and usability of charts in web applications. This makes D3 a natural choice for producing rich, interactive visual representations of structured data.

4.3 SVG

One implementation of the `StringChartGenerator` class targets the generation of stand-alone SVG files. This variant is designed specifically to produce clean SVG output, free of any interactive elements or HTML-specific tags. It focuses solely on encoding the graphical content in standard SVG format, making it suitable for static use cases such as printing, embedding in documents, etc. To ensure readability and maintainability, the resulting SVG markup is formatted using an XML formatting package, which organises the structure of the file with proper indentation and tag hierarchy.

4.4 Canvas

An additional implementation of the `StringChartGenerator` class exists for rendering charts using the HTML `<canvas>` element. This version is capable of generating JavaScript code that draws directly onto a canvas context, offering efficient, low-level rendering capabilities for dynamic or performance-sensitive applications. However, this implementation is currently not in active use, since the D3-based approach has better support for structured SVG generation and interactive features such as mouse hover handling.

Chapter 5

Future Work

During the short time frame of the project of around six weeks, many known problems were fixed and new features implemented. Of course, numerous ideas for improvements and additional features could not be implemented in this time frame and are documented here as ideas for future work.

5.1 Three.js

Using three.js for rendering String Charts would offer some benefits over traditional SVG-based rendering:

- *Better Performance*: Since Three.js leverages GPU hardware accceleration, it can render thousands of graphical elements more efficiently than SVG, which is constrained by DOM manipulation and reflow.
- *Support for 3D Effects*: Three.js enables the use of depth, perspective, and 3D interaction, which can enhance the expressiveness of visualisations involving spatial or overlapping relationships.

Despite its capabilities, Three.js comes with some trade-offs compared to SVG:

- *Increased Complexity*: Setting up a Three.js environment and rendering pipeline is more involved than using declarative SVG markup.
- *Limited Accessibility and Native Interactivity*: Since WebGL content is rendered in a canvas and not part of the DOM, accessibility features and simple event handling require additional logic, unlike in SVG.

Due to the modular structure of the project, integrating Three.js for rendering should be possible with minimal overhead. Specifically, the `StringChartGenerator` base class provides a well-defined interface, and implementing a custom generator using Three.js would only require overriding a few core methods. This abstraction significantly reduces complexity by isolating rendering logic, allowing developers to leverage three.js for high-performance visuals without altering the broader application logic.

5.2 Station/Time Filtering

Currently, by default, all stations and the complete time frame are shown in the string chart. A meaningful extension could be to show only a specific time frame and only a subset of the stations. Both could be done with a double-edged slider. In that context, it could also be useful to have checkboxes beside the stations so a user can tick single stations to be shown or not shown, for example, when stations are close to each other and could overlap each other.

5.3 Trip Filtering

A planned future improvement involves implementing filtering capabilities for GTFS data based on weekdays, weekends, and holidays. Additionally, the system might support distinguishing between incoming and outgoing train trips. These features would enhance the clarity and usability of string chart visualisations by allowing users to focus on specific service patterns and directions of travel, facilitating more detailed transit analysis.

5.4 Flip Vertical/Horizontal

An additional feature would allow users to flip the ordering of start and end stations, as well as the corresponding start and end times of train trips. This functionality would provide greater flexibility in visualising and analysing string charts from different directional perspectives, making the tool more adaptable to various use cases and user preferences.

5.5 Route Styling

The route styling could be improved by adding more parameters for the route lines and stop circles. Currently, colour, stroke width, and stroke pattern can be set. This could be extended with transparency settings, the option to show only one trips towards one direction, and the option to distinguish routes in the opposite direction by their colour. In addition, the styling of lines and circles can be reworked with SVG classes, where the styling parameters are defined and only the class name is included in the inline definition.

5.6 GitHub Workflow - Tauri Releases

It is possible to create a GitHub Actions workflow to automatically build Tauri native packages for Linux, Windows, and macOS [GitHub 2025]. A YAML configuration file (e.g., `.github/workflows/tauri-build.yml`) is created using a matrix strategy. This matrix uses the `runs-on` key to define a set of operating systems (`ubuntu-latest`, `windows-latest`, and `macos-latest`), ensuring that the job runs on each corresponding virtual environment. Each job step typically involves checking out the code, setting up the Node.js and Rust toolchains, and installing platform-specific dependencies when needed.

The core build process relies on the official `tauri-apps/tauri-action`, which compiles the app using the Tauri CLI. It also optionally handles code signing (if secrets are configured), creates GitHub Releases, and generates updater metadata such as `latest.json`, allowing Tauri apps to support seamless cross-platform updates [Tauri 2025b]. This automated setup streamlines the packaging and distribution of desktop applications across major operating systems.

5.7 Data Editing

Tools like jTrainGraph [Scherzinger 2022] offer the opportunity to edit routes and times. This also pays tribute to one of the origins of string charts: To use it not only as a representation, but also as a planning tool. It should then also be possible to save the edited data in GTFS format.

5.8 Chart Export Formats

Besides the existing SVG export, exporting to raster graphics formats such as PNG would be possible.

Appendix A

General Transit Feed Specification (GTFS)

The General Transit Feed Specification (GTFS) is an open standard for describing public transport schedules [GTFS 2025]. It was initially developed through collaboration between Google and TriMet, the public transport agency in Portland, Oregon, with the purpose to have standardized API for schedules and geographical data. GTFS has been widely adopted by transport authorities, developers, and researchers worldwide.

GTFS consists of a structured collection of plain text files, typically packaged as a ZIP archive. Each file contains comma-separated values and represents a particular aspect of a transit system, such as agencies, stops, routes, trips, and service frequencies.

The default structure of GTFS Static has to contain at least the following text files:

- `agency.txt`
- `stops.txt`
- `routes.txt`
- `trips.txt`
- `stop_times.txt`
- `calendar.txt`

The widespread distribution of GTFS improved the accessibility and interoperability of transport data. A common format makes it possible for developers to integrate transit information into applications and services, for example in routing applications such as Google Maps. It is also used in traffic research for example in network coverage, transit needs, and multimodal connectivity [Google 2025].

An extension to the static format is GTFS Realtime, which allows transport companies to provide dynamic updates like delays, vehicle positions or deviations.

The popularity of GTFS shows the need for interoperability and connectivity through different transport agencies and enterprises and routing services and apps. Open data supports this trend toward a democratic and easy accessible solutions. Some publicly available GTFS datasets include:

- *ÖBB*: Austrian Railways offer yearly updated datasets of their train services [ÖBB 2025].
- *Mobilitätsverbünde Österreich*: The Mobility Association Austria is an association of seven regional public transport authorities, offering GTFS datasets [MAA 2025]. Registration is required to agree to the terms of use.
- *GoNortheast*: The data of this British transport operator is available under the Open Government License Version 3.0 [GoNorthEast 2025].

- *GTFS.de*: This private German website collects GTFS data for several public transport modes in Germany and provides them under a CC BY 4.0 license [GTFS.de 2025].

Bibliography

Barry, Mike and Brian Card [2014]. *Visualizing MBTA Data*. 10 Jun 2014. https://mbtaviz.github.io/ (cited on page 1).

Bostock, Michael [2025]. *D3*. 26 Jun 2025. https://d3js.org/ (cited on page 11).

GitHub [2025]. *Workflow syntax for GitHub Actions*. 26 Jun 2025. https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions (cited on page 14).

GoNorthEast [2025]. *Network Data*. Go North East, 2025. https://gonortheast.co.uk/open-data (cited on page 15).

Google [2025]. *GTFS Static Overview*. 12 May 2025. https://developers.google.com/transit/gtfs (cited on page 15).

Gsellmann, Inge, Michael Hebesberger and Danijela Lazarevic [2023]. *String Charter 2: Visual Transport Schedules*. 03 Jul 2023. https://courses.isds.tugraz.at/ivis/projects/ss2023/ivis-ss2023-g2-project-string-charter2.pdf (cited on page 1).

GTFS [2025]. *General Transit Feed Specification (GTFS)*. 12 May 2025. https://gtfs.org/ (cited on page 15).

GTFS.de [2025]. *Deutschlandweite GTFS Feeds*. 2025. https://gtfs.de/de/feeds/ (cited on page 16).

Gulp [2025]. *Gulp - The streaming build system*. 26 Jun 2025. https://gulpjs.com/ (cited on page 4).

MAA [2025]. Mobility Association Austria, 12 May 2025. https://data.mobilitaetsverbuende.at/en (cited on page 15).

MDN [2025]. *Window.showSaveFilePicker() - Web APIs | MDN*. Mozilla Developer Network, 26 Jun 2025. https://developer.mozilla.org/en-US/docs/Web/API/Window/showSaveFilePicker (cited on page 7).

Nepal, Trinish, Martin Rabensteiner and Stephan Robinig [2025]. *String Charts: Visual Transport Schedules*. 07 May 2025. https://courses.isds.tugraz.at/ivis/surveys/ss2025/ivis-ss2025-g3-survey-string-charts.pdf (cited on page 1).

ÖBB [2025]. *Soll Fahrplan GTFS*. 2025. https://data.oebb.at/datensaetze~soll-fahrplan-gtfs~ (cited on page 15).

Robinig, Stephan, Martin Rabensteiner, Inge Gsellmann, Michael Hebesberger and Danijela Lazarevic [2025a]. *String Charter 3 Demo*. 26 Jun 2025. https://stofflr.github.io/String-Charter-3/ (cited on page 1).

Robinig, Stephan, Martin Rabensteiner, Inge Gsellmann, Michael Hebesberger and Danijela Lazarevic [2025b]. *String Charter 3 Repository*. 26 Jun 2025. https://github.com/StofflR/String-Charter-3/ (cited on page 1).

Scherzinger, Moritz [2022]. *jTrainGraph*. 30 Dec 2022. https://jtraingraph.de/ (cited on pages 1, 14).

Tailwind [2025]. *Tailwind CSS: A Utility-First CSS Framework*. Tailwind Labs, 26 Jun 2025. `https://tailwindcss.com/` (cited on page 4).

Tauri [2025a]. *Tauri*. 11 May 2025. `https://tauri.app/` (cited on page 3).

Tauri [2025b]. *Tauri GitHub Pipeline*. 26 Jun 2025. `https://v2.tauri.app/distribute/pipelines/github/` (cited on page 14).

Vue.js [2025]. *Vue*. 26 Jun 2025. `https://vuejs.org/` (cited on page 3).

Yarn [2025]. *Yarn Package Manager*. 26 Jun 2025. `https://yarnpkg.com/` (cited on page 3).