Data Visualization Design Systems

Laura Thaçi, Bastian Kandlbauer, Laura Pessl, and Vera Tysheva

DAT.C406UF Information Visualisation SS 2025
Graz University of Technology

12 May 2025

Abstract

This survey explores data visualization design systems, starting with an introduction to design systems more generally. A design system is a set of standardized styles, guidelines, and reusable components aimed at providing a shared language and visual consistency across a product or organization. A data visualization design system provides such guidance for for charts, graphs and other visual representations of data. To provide a better feeling of how such data visualization design systems work in practice, several tools and examples were analyzed and discussed. Additionally, a prototype data visualization design system was proposed for Graz University of Technology and documented using Storybook.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By providing structured and reusable components, that promote consistency, efficiency and scalability across various teams and platforms, design systems have become a major part of modern software development. They encompass a wide range of elements, from components and interaction patterns to accessibility standards, content guidelines and governance rules. Data visualization design systems address specific challenges of representing data in a clear, consistent and meaningful way. This survey provides resources to different design system collections and a detailed description about selected tools. To illustrate the concepts in a data visualization design system, a custom data visualization design system was proposed for Graz University of Technology and was documented with Storybook [Chromatic 2025].

Chapter 2

Design Systems

As digital products become more complex and design teams have to become more versatile, the need for consistent, reusable and maintainable design solutions has never been greater. To address these challenges, design systems were developed to serve as structured frameworks, which align a visual language, interaction patterns and development best practices. Design systems serve as comprehensive toolkits, which enable teams and organizations the creation of consistent user interfaces through their different applications. By combining visual design specifications, coded components and documentations, they support unified software development across different platforms and scopes.

Far from being static repositories, modern design systems are more comparable with dynamic ecosystems. They enable shared ownership design standards, encourage the reuse of elements and facilitate collaboration across disciplines. Design systems centralize the guidelines for design and development, helping to reduce redundancy and to minimize inconsistency [IDF 2021].

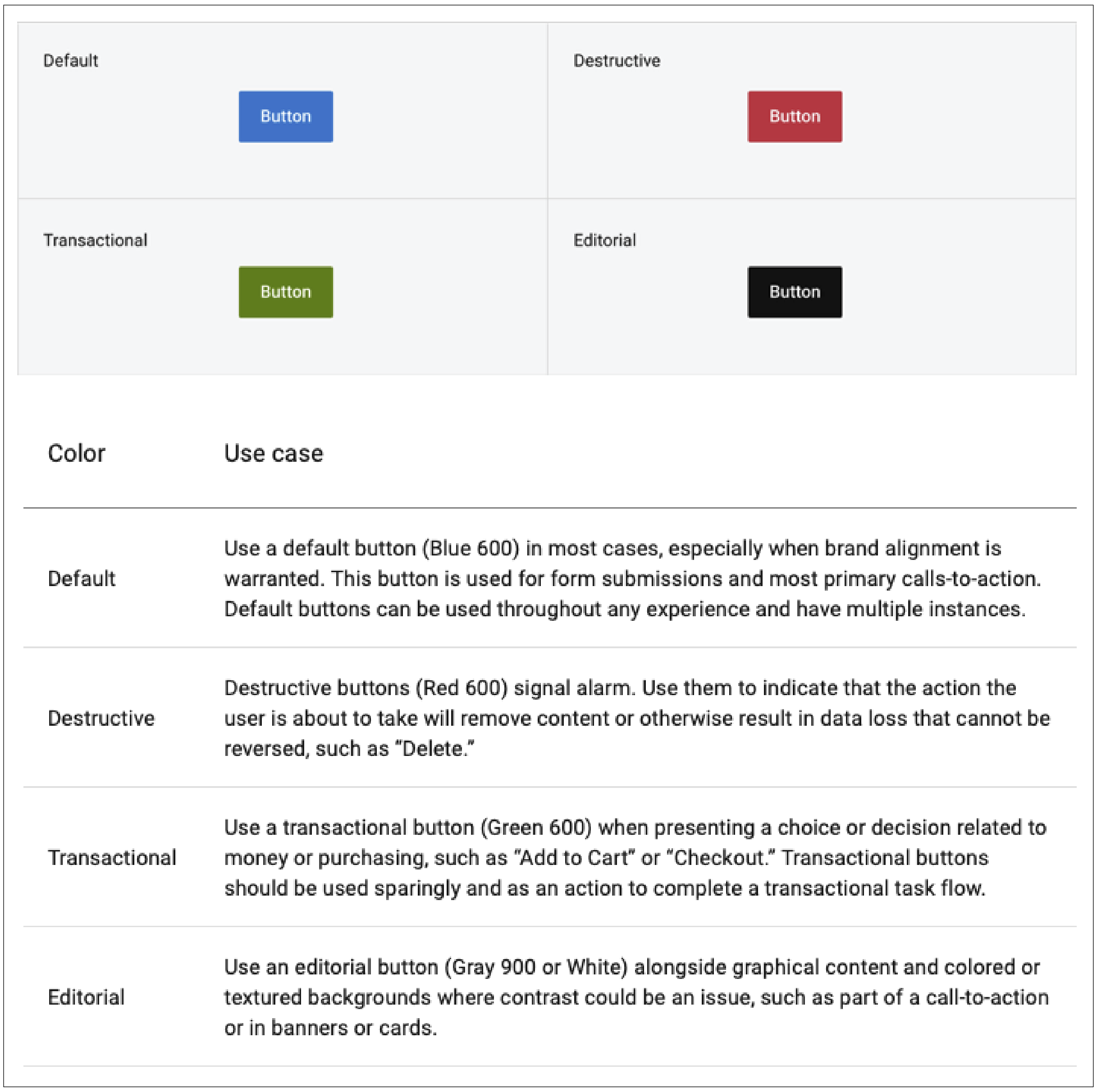
2.1 Core Elements of a Design System

Design systems are composed of several interrelated components. Following the discussion by Callahan [2022], the most important components are:

- *Visual and interaction guidelines.*
- *Design tokens.*
- *Reusable components.*
- *Documentation and usage guidelines.*
- *Governance and contribution models.*

2.1.1 Visual and Interaction Guidelines

A strong visual identity begins with comprehensive visual and interaction guidelines. The aim of these guidelines is to provide the aesthetic and behavioral foundations for interfaces and define how elements appear and respond to user input. Visual guidelines cover different aspects, such as color palettes, typography, spacing, iconography or grid systems. They help to establish a recognizable brand identity and ensure visual harmony across all applications within an organization. Interaction guidelines focus on different dynamic behaviors - how buttons animate on hover, how error messages are displayed or how navigation elements respond to touch or keyboard input. Combining these rules, they support both, usability and accessibility. Since these guidelines are often the most referenced part of a design system, they serve as the foundation for consistent, user-friendly interfaces.



Color	Use case
Default	Use a default button (Blue 600) in most cases, especially when brand alignment is warranted. This button is used for form submissions and most primary calls-to-action. Default buttons can be used throughout any experience and have multiple instances.
Destructive	Destructive buttons (Red 600) signal alarm. Use them to indicate that the action the user is about to take will remove content or otherwise result in data loss that cannot be reversed, such as "Delete."
Transactional	Use a transactional button (Green 600) when presenting a choice or decision related to money or purchasing, such as "Add to Cart" or "Checkout." Transactional buttons should be used sparingly and as an action to complete a transactional task flow.
Editorial	Use an editorial button (Gray 900 or White) alongside graphical content and colored or textured backgrounds where contrast could be an issue, such as part of a call-to-action or in banners or cards.

Figure 2.1: Dell design system: button colors [Dell 2025b]. [Screenshot taken by Bastian Kandlbauer.]

2.1.2 Design Tokens

The next layer of a design system is the layer of design tokens, the smallest multi-platform units of style definition. These tokens represent corporate design rules as semantically named variables, each representing a single style such as a color, font size, font family or margin. Instead of hard coding these values multiple times into individual style sheets of different components, they are only defined once in a central configuration file and then referenced throughout the system.

The use of design tokens supports the overall structure and consistency of a design implementation. For example, when coloring a button, the color is not applied directly as:

```
blue-600
```

but is instead referenced using design tokens such as:

```
--default: blue-600;
--destructive: red-600;
--transactional: green-600;
```

```
    --editorial: gray-900;
```

Examples can be found in the Dell Design System [Dell 2025b], such as for the Button component shown in Figure 2.1.

If any of those defined styles needs to be changed, updating the token within the configuration file results in system-wide changes for all components that rely on it. This abstraction not only enables theming and accessibility, for example switching to dark mode by redefining a color token, but also facilitates consistent implementations across various platforms such as web, iOS and android. Since these tokens serve as a single source of truth, they form the technical and visual foundation on which the rest of the system is built.

2.1.3 Reusable Components

Another key feature of design systems are libraries of reusable components. Reusable components are standardized interface elements, which are implemented in both design and code and represent different building blocks, such as buttons, input-fields, check-boxes, modals or tool-tips. Each of these individual components is documented in detail, including its structure, states, use cases and best practice implementation examples. Many design systems also provide an interactive preview or other environments, where developers can explore the components.

The components themselves are built around a modular structure, enabling organizations to assemble complex user interfaces quickly and reliably. Since they are centrally maintained, any updates or bug fixes can propagate automatically across all their instances. The use of reusable components reduces technical debt and encourages the adherence of design standards across an organization and its different teams.

2.1.4 Documentation and Usage Guidelines

For a design system to be effective, its components and principles must be clearly documented. Without any documentation, developers of organizations are left guessing about intended usage, thereby reducing consistency and maintainability. Well presented documentation serves as both a reference and as a teaching tool, enabling consistent applications and easier integration for new designers and developers.

A design system's documentation typically includes:

- *Visual and interaction examples*: Illustrations or demos, which show the components appearance and behavior through various states, for example in default state, in hover state or in error state.
- *Best practices and guidelines*: Practical recommendations, which help designers and developers to apply components or visualizations correctly and consistently, based on their intended use cases and known constraints.
- *Implementation instructions*: Code snippets, examples of integrations and technical details, which are necessary for incorporating components into different environments.
- *Accessibility notes*: Annotations on how components support inclusive design, such as focus indicators or color contrast compliance.
- *Design and code references*: Links to relevant design assets, such as component libraries or Storybook documentation.
- *Versioning and change logs*: Records of changes, including updates and bug fixes, which help to keep track of evolution.

Comprehensive documentation minimizes uncertainty and enhances the communication across different teams of an organization. By promoting the long-term usage and contribution, it plays a vital role in driving the adoption and durability of a design system.

2.1.5 Governance and Contribution Models

Design systems tend to grow over time. Without any proper governance, they risk becoming disorganized, inconsistent or overly complex. A governance model outlines how decisions are made, who is allowed to contribute and how different changes are approved. This governance structure may include, for example:

- Roles such as maintainers, reviewers or design leads.
- Workflows for proposing, reviewing and finally merging changes.
- Policies for versioning of different components.

An effective governance strategy supports an organization’s long-term scalability and ensures quality control. It enables the system to adapt to new standards, without compromising stability. In some organizations, dedicated DesignOps teams are responsible for handling governance and supporting internal tooling around the system [Ramamohan 2023].

2.2 Style Guides, Pattern Libraries and Design Systems

The terminology in design infrastructure can often be ambiguous and confusing. While the terms “style guide”, “pattern library” and “design system” are often used in the same context, they sometimes differ in their scope and purpose. Following the discussion by Andrews [2025]:

- *Style Guide*: Focuses on perceptual patterns, such as colors or fonts.
- *Pattern Library*: A more general tool to capture, collect and share design patterns and guidelines for their usage, like components or text fields.
- *Design System*: A set of interconnected patterns and shared practices, including code snippets and widget libraries.

This layered evolution, starting from simple visual guidelines to fully-integrated systems, represents the maturing of design practices within digital product development.

2.3 Benefits of Design Systems

The main benefit of design systems is that they ensure general uniformity across an organization and through its different applications. Through the standardization of components, styles and interaction patterns, the resulting digital products appear more unified. This consistency not only enhances the systems overall user experiences, but also improves an organizations brand identity, which is responsible for building trust with users.

Another key benefit of design systems is increased efficiency. By reusing the systems defined components and design tokens, developers avoid the effort of duplication in design and development. Instead of repeatedly solving the same problems, they can focus on higher-level challenges, while knowing that the foundational elements of their created applications are already established and tested. Through this workflow, the product development cycle is significantly accelerated.

Design systems also promote scalability. Since organizations tend to grow and different teams are established, a central system ensures that all contributors are working from the same shared set of standards, making it also easier to onboard new team members and expand the product offerings. By providing a shared language and a clear documentation, the gaps and miscommunication between design, development and product managements are decreased.

For these reasons, a well-maintained design system contributes not only to better products but also to a healthier and more productive workflow through an organization [Mir 2023].

2.4 Challenges of Design Systems

While design systems offer several important benefits, also some challenges are introduced, which complicate the development and long-term maintenance of such systems. One of the most significant challenges is the initial investment, needed for the creation of a design system.

The establishment of a solid foundation of design tokens, reusable components, documentation and governance processes requires time and resources. Since immediate deliverables often take priority within organizations, those demands can often be challenging. Even when a design system is established, achieving the organization-wide adoption to it is not guaranteed. Since designers and developers often want to stick to their existing workflow or might resist constraints that appear to limit creativity, a design system may fail to gain traction, resulting in fragmented implementations and reduced effectiveness.

A design system is never finished. This brings up the challenge of ongoing maintenance, including the updating of components, refining guidelines and managing the dependencies across versions. While the purpose of a design system is to provide consistency, it must still allow for contextual variation and innovation [UXPin 2023].

Chapter 3

Data Visualization Design Systems

A data visualization design system is a structured framework that specifies how data visualizations should be designed, built, and used aligned with brand standards. Usually part of a larger design system, it provides guidelines, reusable components and best practices for charts, graphs, maps and other data representations to ensure clarity and accessibility. It standardizes color usage, layout, typography and other data visualizations design elements, as can be seen in Figure 3.1 for the Dell Design System.

3.1 Importance

A well-defined data visualization design system addresses not only visual consistency, but also considers user interaction patterns and accessibility requirements [Greben 2025a]. It helps create cohesive, scalable, and effective data-driven charts and interfaces:

- *Scalability*: Re-usability of components allows the rapid development and deployment of data visualizations across multiple different projects and platforms.
- *Consistency*: A standardized approach to data representation ensuring consistency and uniformity in the use of visual elements reduces cognitive load from the users allowing them to focus directly on the data insights.
- *Efficiency*: Predefined guidelines and components help to reduce design time as well as facilitate collaboration between designers and developers by providing a common visual language.
- *Accessibility*: Accessibility guidelines ensure that data visualization is comprehensible to all users including those with visual impairments.
- *User Engagement*: Interactive data visualizations that include tooltips, hover states and other interactive ways enhance user engagement by presenting data in an intuitive manner.

3.2 Core Concepts

The core concepts described in this section act as the foundation for creating clear and cohesive data visualizations. The principles include the use of color, typography and recommendations for data structuring based on the examples collected by Cesal [2019]:

- *Design Principles*: Fundamental guidelines that align with the brand's visual identity and user experience goals. Principles may include simplicity, clarity, visual hierarchy, and consistency. A standardized set of design principles acts as the foundation for creating effective visualizations that present data accurately and efficiently.
- *Color Palette*: A defined set of colors for different data types and categories, emphasizing accessibility and visual clarity. Colors are specified for primary data points, secondary data points,

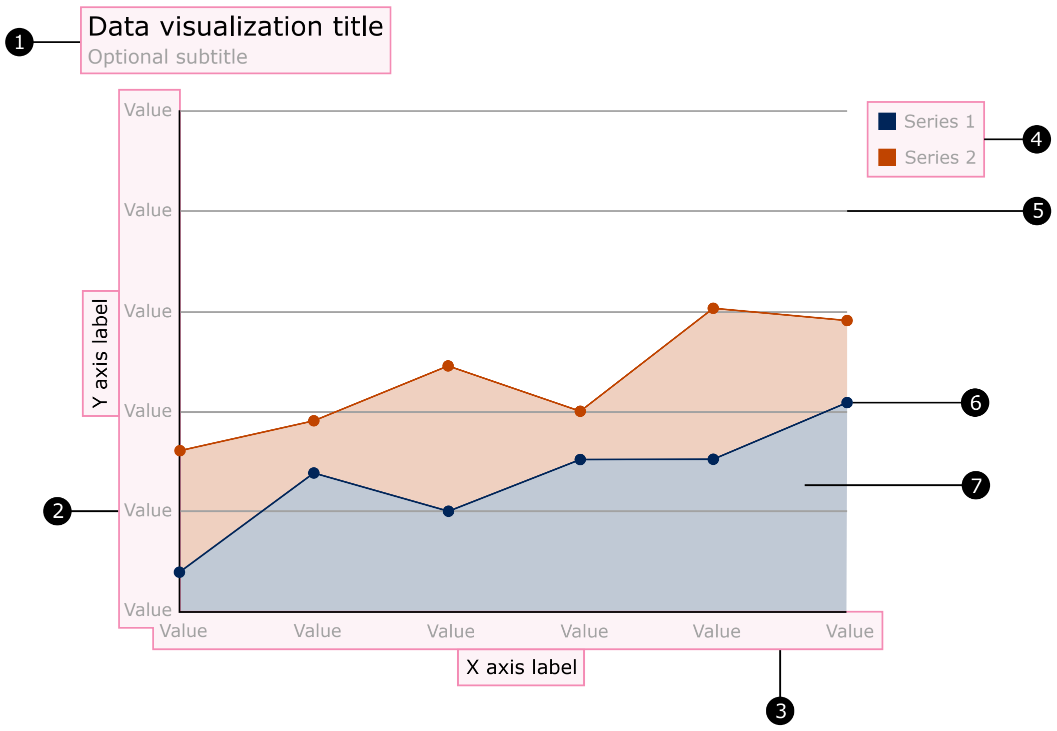


Figure 3.1: Dell Design System: stacked area chart. [Redrawn by Bastian Kandlbauer from the original by Dell [2025d].]

background elements, and alerts. Gradients and data heatmaps can also be included. Additionally, it is important to add color-blind-friendly palettes and ensure sufficient contrast for better readability.

- *Chart Types*: A library of pre-defined visualizations (such as bar charts, line graphs, and heat maps) with designated use cases. Each chart type includes guidelines for when and how to use it, data formatting, and interaction patterns. Includes best practices for displaying complex datasets to manage proper information density.
- *Typography*: Specified font styles, sizes, and hierarchy for titles, labels, data points, and annotations. Font guidelines ensure readability across all screen sizes and devices as well as maintain brand consistency.
- *Interaction Patterns*: Standardized guidelines for user interactions with data visualizations, such as hover effects, tooltips, zooming, panning, and data filtering. Interaction patterns also cover error handling and loading states. Outlined best practices for user feedback including visual cues for data loading and contextual information on hover.
- *Data Components Library*: Reusable components (e.g. legends, axes, data points, labels, and data grids) that maintain a consistent visual look and feel. Each component is documented with specifications for size, spacing, and interaction behavior.
- *Accessibility Standards*: WCAG (Web Content Accessibility Guidelines) standards for color contrast, keyboard navigation, screen reader compatibility, and responsive design. Accessibility guidelines also cover text alternatives for visual elements and ensuring that visualizations remain legible under various conditions.

3.3 Best Practices

According to Czaban [2023], best practices for creating effective data visualizations include:

- *Clarity and Simplicity*: The use of minimal visual elements and focus on essential data points to prevent information overload and guide user focus to critical data insights.
- *Documentation*: Comprehensive documentation to cover design principles, component libraries, interaction guidelines, and accessibility standards.
- *Responsiveness*: Data visualizations are adaptable to different screen sizes and devices such as mobile, desktop and tablet.
- *Testing*: The effectiveness of visual components is continuously evaluated through user feedback, usability testing, and performance monitoring.

Chapter 4

Examples of Data Visualization Design Systems

Many individuals and organizations have developed and published data visualization design systems. The Design Systems Database is an online collection created by Greben [2025b], featuring 72 design systems, of which 23 specifically focus on data visualization. Each entry includes a detailed description of its components, along with links to additional resources such as the originator's web site and code repository. The web site also provides useful filtering mechanisms to easily identify design systems that, for example, provide a Storybook [Chromatic 2025] interface to interactively explore its elements.

The Adele repository includes only publicly available design systems and is maintained by Treder [2025]. The collection is organized in a well-structured table that contains key details such as the author, the framework used, and links to the code repository. This extensive collection encompasses over 100 entries and continues to grow as users are encouraged to contribute their own or any other public design system to the list via the Adele repository [UXPin 2025].

4.1 Open-Source Data Visualization Design Systems

Three open-source data visualization design systems were selected for closer investigation: PatternFly, PatternFly, and Carbon. For each of the three chosen design systems, the bar chart component was evaluated using the Graz Population dataset shown in Table 4.1. An overview of their characteristics can be found in Table 4.2.

4.1.1 PatternFly

PatternFly was created by Red Hat [2025b] and is an open-source design system that provides a high number of data visualization components. The system uses the React library [Meta 2025] and can be used in local projects by following the instructions on the public Github repository [Red Hat 2025a]. The web site incorporates extensive documentation and examples of area, bar and line charts, as well as other chart types such as box plot and scatter plot. To interactively try out the visualizations, a JavaScript [MDN 2025] code box can be opened directly underneath each chart. Alternatively, a link to use CodeSandbox [CodeSandbox 2025] to edit the charts is provided, which however had difficulties handling the editing of the code and can therefore not be recommended.

In addition, PatternFly includes a comprehensive description about the usage of colors for data visualization. The system provides multiple color palettes which can be used by selecting the corresponding design token in the code and defining the number of different colors needed. While legends are typically static elements used to display chart information, tooltips are an effective way of revealing detailed data values upon hovering over a chart. PatternFly offers various types of legends and tooltips, which were

Year	Population
2024	302,749
2023	298,479
2022	292,630
2021	291,134
2020	291,072
2019	288,806
2018	286,292

Table 4.1: Population of the city of Graz, on 01 Jan of each year from 2018 to 2024 [Land Steiermark 2025].

	PatternFly	**Pajamas**	**Carbon**
Originator:	Red Hat	GitLab	IBM
License:	MIT	MIT	Apache 2.0
Chart Types:	Area, Bar, Box Plot, Bullet, Donut, Line, Pie, Scatter Plot	Area, Bar, Gauge, Line, Scatter Plot, Sparkline	Alluvial, Area, Bar, Box Plot, Bubble, Donut, Floating Bar, Gauge, GeoMap, Heatmap, Histogram, Line, Lollipop, Network, Pie, Parallel Coordinates, Radar, Scatter Plot, Stream, Treemap, Wordcloud
Documentation:	`https://patternfly.org/`, CodeSandbox	`https://design.gitlab.com/data-visualization/overview`, `https://gitlab-org.gitlab.io/gitlab-ui/`	`https://carbondesignsystem.com/data-visualization/getting-started/`
Distribution:	`https://github.com/patternfly/patternfly-org`	`https://figma.com/file/...`, `https://gitlab.com/gitlab-org/gitlab-services/design.gitlab.com`	`https://github.com/carbon-design-system/carbon`
Framework:	React	Vue	Angular, React, Svelte

Table 4.2: Overview of three selected open-source data visualization design systems.

implemented using the Victory library [Nearform 2025]. A built-in SVG download for charts is currently not provided by the system, and would have to be implemented separately by the user.

To evaluate the system, a simple bar chart was created using the Graz population dataset, as shown in Figure 4.1. The range of the y-axis must be set manually, while tooltips for the bars are created automatically upon the chart's creation. A showcase video demonstrating the editing process, as well as the PatternFly documentation web site, was created by Pessl [2025c].

4.1.2 Pajamas

The open-source design system Pajamas by GitLab [2025a] includes several chart types, such as area, bar and gauge charts. The system uses the Vue framework [You 2025] and provides a Storybook interface [GitLab 2025c] to easily adapt values and generate custom data visualizations. Additionally, the source code can be observed by downloading code from a public GitLab repository [GitLab 2025b]. Charts

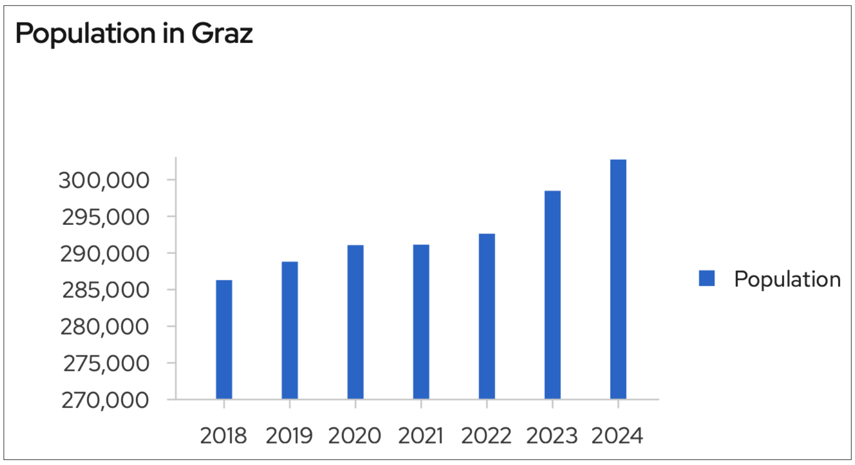


Figure 4.1: PatternFly: Bar chart showing the population of Graz. [Screenshot created by Laura Pessl.]

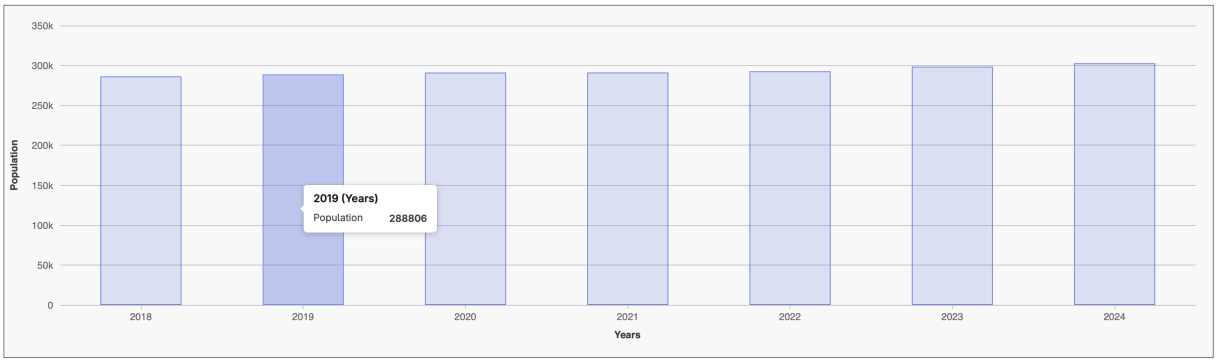


Figure 4.2: Pajamas: Bar chart showing the population of Graz. [Screenshot created by Laura Pessl.]

created using Pajamas can be enhanced using a specific data visualization color palette that includes color sequences designed for sequential, categorical and divergent data types. The system also provides a variety of tooltip designs. Unfortunately, the Pajamas Storybook does not provide a built-in SVG download for charts.

Figure 4.2 shows a bar chart of the Graz population dataset, which was created using the Storybook interface. Pajama's bar charts need a selection of the x-axis type which can be set to value, category, time or log. The y-axis is automatically scaled, based on the provided dataset and inherently starts at zero. Furthermore, tooltips are generated by Pajamas, which can be revealed by hovering over the bars. For a better understanding of the documentation web site and how the Pajamas Storybook can be used, a showcase video [Pessl 2025b] is available.

4.1.3 Carbon

The Carbon data visualization design system is published by IBM [2025a] and encompasses a large number of different chart types. Besides commonly used area, bar and line charts, this system also allows the creation of more complex charts such as word clouds, tree maps and geographical charts. The frameworks Angular [Google 2025], React [Meta 2025], and Svelte [Svelte 2025] are all supported. Since the design system is open-source, a GitHub repository is publicly available [IBM 2025b]. On the Carbon web site, there are comprehensive explanations and multiple examples on how and why to use axes and labels as well as legends and tooltips. The system additionally provides color palettes for categorical and sequential data.

A bar chart of the Graz population dataset was created using the provided CodeSandbox interface, and

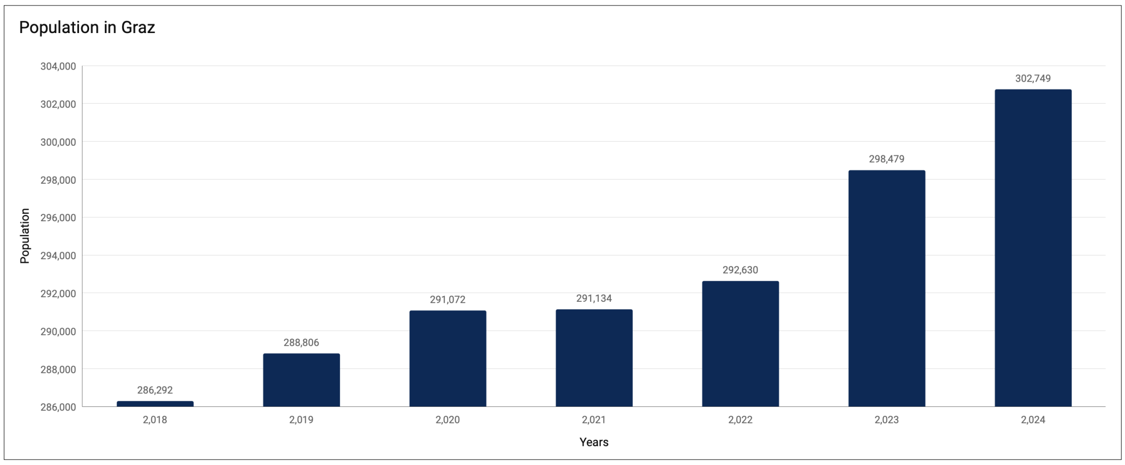


Figure 4.3: Carbon: Bar chart showing the population of Graz. [Screenshot created by Laura Pessl.]

is shown in Figure 4.3. Although direct export of the chart as an SVG file is not supported, CodeSandbox provides functionality to embed the chart using an iFrame. A showcase video about the Carbon data visualization design system is available [Pessl 2025a].

4.2 Proprietary Data Visualization Design Systems

Three proprietary data visualization design systems were also looked at: Dell Design System, Druids, and Motion. An overview of their characteristics can be found in Table 4.3.

4.2.1 Dell Design System

The Dell Design System is a proprietary system developed by Dell [Dell 2025a]. Although its documentation and associated Storybook [Dell 2025c] are publicly accessible, the full source code and distribution of the system are not freely available. Nevertheless, the system offers a wide range of chart examples across popular frameworks including Angular, React, Vue and Svelte.

4.2.2 Druids

Druids is a closed-source design system published by DataDog [DataDog 2025]. It focuses on complex data visualizations, including funnel charts, geomaps, and topology maps. The web site provides only static screenshots of these charts, as the underlying React components are part of DataDog's commercial product. Nonetheless, the system offers detailed guidelines on color usage and tooltip design.

4.2.3 Motion

Motion is a proprietary design system developed by Michelin [Michelin 2025]. It supports a variety of chart types such as bar, bubble, pie and line charts. High-level guidance about the usage of charts as well as detailed descriptions of individual chart types can be found on the web site. Although a private GitHub repository for the Angular, Blazor [Microsoft 2025a] and Svelte frameworks is referenced, access is restricted to Michelin employees. However, the publicly available documentation offers valuable insights into the design considerations behind the system's data visualizations.

	Dell	**Druids**	**Motion**
Originator:	Dell	DataDog	Michelin
License:	Proprietary	Proprietary	Proprietary
Chart Types:	Area, Bar, Bubble, Donut, Gauge, Line, Metrics Card, Pie	Pie, Funnel, GeoMap, Heatmap, Histogram, Query Value, Scatter Plot, Timeseries, Topology Map, Treelist	Bar, Bubble, Donut, Line, Pie, Polar, Radar, Scatter Plot, Treemap, Waterfall
Documentation:	`https://delldesignsystem.com/data-visualization/ddv-overview/`, `https://vanilla-dv.delldesignsystem.com/`	`https://druids.datadoghq.com/patterns/dataviz`	`https://designsystem.michelin.com/data-visualization/introduction`
Distribution:	`https://figma.com/design/...`	Internal Figma Kit	Private GitHub
Framework:	Angular, React, Vue, Svelte	React	Angular, Blazor, Svelte

Table 4.3: Overview of three selected proprietary data visualization design systems.

Chapter 5

TU Graz Data Visualization Design System (Storybook)

This chapter describes the creation of a proposed data visualization design system for Graz University of Technology (TU Graz), based on the university's corporate design guidelines [TU Graz 2019]. In this project, the university is referred to by its German name: TU Graz. The aim was to create a set of reusable React components for visualizing data, while adhering to the TU Graz corporate design rules. Storybook was used as the main tool for showing, testing, and documenting these components. The idea was to create a clear visual style for data charts for TU Graz projects, making it easier for developers to create charts and visuals that are informative and match the TU Graz brand.

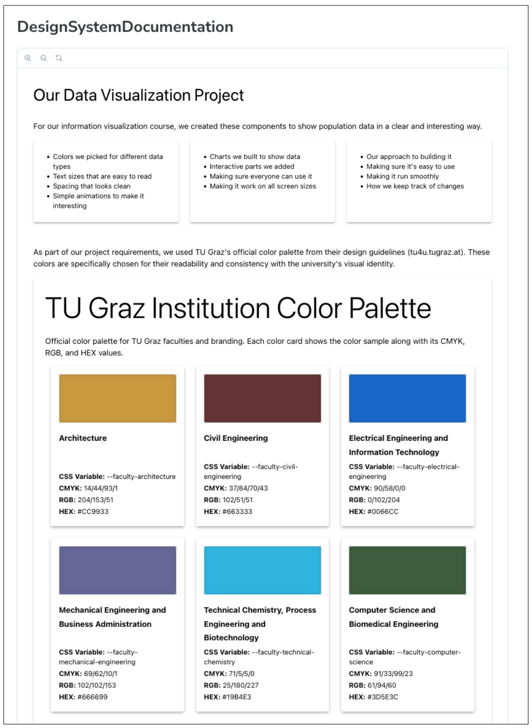
5.1 Project Setup and Core Technologies

The project started as a regular React application. TypeScript was used to help with code quality by adding static typing. React (v18.x) was used for building the user interface components [Meta 2025]. TypeScript (v5.x) adds static type checking to JavaScript code [Microsoft 2025b]. Storybook (v8.x) served as the main tool for working on and showing components interactively [Chromatic 2025]. For the actual visualizations, Recharts and Chart.js (via `react-chartjs-2`) were used [Recharts 2025]. Recharts was selected for some charts due to its straightforward nature, while Chart.js was used when more flexibility was needed. Material-UI (MUI) was incorporated for UI parts in documentation components and potentially for parts of the charts themselves [MUI 2025].

5.2 Following TU Graz Design Guidelines

Ensuring all charts and the Storybook setup matched the official TU Graz design rules was crucial. A central theme file, `src/styles/colors.css`, was set up to hold all main color definitions. This file contains clear definitions of main, secondary, and faculty-specific colors based on TU Graz branding. These design tokens were made available as CSS custom properties (global variables), as shown in Listing 5.1. They can then be re-used in both JavaScript and CSS for faculty colors on chart parts such as like lines in a multi-line chart or bars in a column chart.

The `PopulationColumnChart.tsx` and `MasterStudentsBarChart.tsx` components use the `react-chartjs-2` library, which is a React wrapper for Chart.js. It was crucial to specifically list all the Chart.js parts needed (controllers, scales, elements) at the top of the component file. This prevents the build process from accidentally removing needed Chart.js code when creating the final version of the project. The population and student data passed in as props is transformed into the format that Chart.js needs for its `data.datasets` and `data.labels`. Settings for axes, tooltips, and legends are also configured in the `options` prop. Colors for the bars and other chart parts come from the global theme file to ensure they match TU Graz branding. Default tooltips were enabled to show data values when hovering over a bar.



DesignSystemDocumentation

Our Data Visualization Project

For our information visualization course, we created these components to show population data in a clear and interesting way.

- Colors we picked for different data types
- Text sizes that are easy to read
- Spacing that looks clean
- Simple animations to make it interesting

- Charts we built to show data
- Interactive parts we added
- Making sure everyone can use it
- Making it work on all screen sizes

- Our approach to building it
- Making sure it's easy to use
- Making it run smoothly
- How we keep track of changes

As part of our project requirements, we used TU Graz's official color palette from their design guidelines (tu4u.tugraz.at). These colors are specifically chosen for their readability and consistency with the university's visual identity.

TU Graz Institution Color Palette

Official color palette for TU Graz faculties and branding. Each color card shows the color sample along with its CMYK, RGB, and HEX values.

Architecture
CSS Variable: --faculty-architecture
CMYK: 14/44/93/1
RGB: 204/153/51
HEX: #CC9933

Civil Engineering
CSS Variable: --faculty-civil-engineering
CMYK: 37/84/70/43
RGB: 102/51/51
HEX: #663333

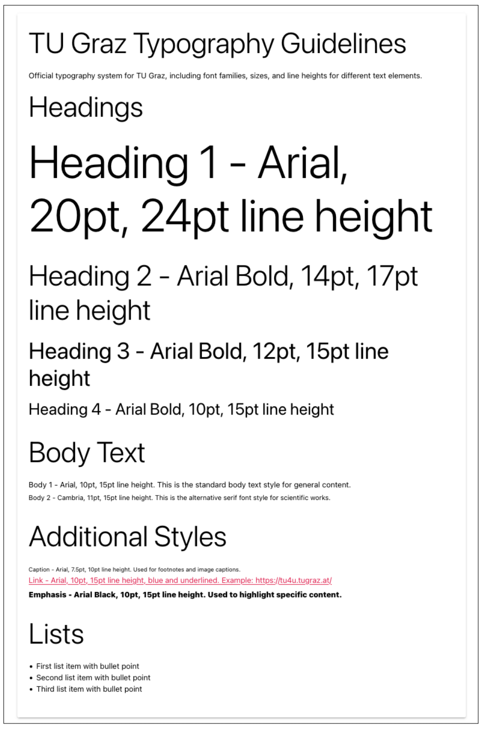
Electrical Engineering and Information Technology
CSS Variable: --faculty-electrical-engineering
CMYK: 90/58/0/0
RGB: 0/102/204
HEX: #0066CC

Mechanical Engineering and Business Administration
CSS Variable: --faculty-mechanical-engineering
CMYK: 69/62/10/1
RGB: 102/102/153
HEX: #666699

Technical Chemistry, Process Engineering and Biotechnology
CSS Variable: --faculty-technical-chemistry
CMYK: 71/5/5/0
RGB: 25/180/227
HEX: #19B4E3

Computer Science and Biomedical Engineering
CSS Variable: --faculty-computer-science
CMYK: 91/33/99/23
RGB: 61/94/60
HEX: #3D5E3C

Figure 5.1: TU Graz Institution [Screenshot created by Laura Thaci.]



TU Graz Typography Guidelines

Official typography system for TU Graz, including font families, sizes, and line heights for different text elements.

Headings

Heading 1 - Arial, 20pt, 24pt line height

Heading 2 - Arial Bold, 14pt, 17pt line height

Heading 3 - Arial Bold, 12pt, 15pt line height

Heading 4 - Arial Bold, 10pt, 15pt line height

Body Text

Body 1 - Arial, 10pt, 15pt line height. This is the standard body text style for general content.
Body 2 - Cambria, 11pt, 15pt line height. This is the alternative serif font style for scientific works.

Additional Styles

Caption - Arial, 7.5pt, 10pt line height. Used for footnotes and image captions.
Link - Arial, 10pt, 15pt line height, blue and underlined. Example: https://tu4u.tugraz.at/
Emphasis - Arial Black, 10pt, 15pt line height. Used to highlight specific content.

Lists

- First list item with bullet point
- Second list item with bullet point
- Third list item with bullet point

Figure 5.2: TU Graz Typography Guidelines [Screenshot created by Laura Thaci.]

```
:root {
  --faculty-architecture: #CC9933;
  --faculty-civil-engineering: #663333;
  --faculty-electrical-engineering: #0066CC;
  --faculty-mechanical-engineering: #666699;
  --faculty-technical-chemistry: #19B4E3;
  --faculty-computer-science: #3D5E3C;
  --faculty-mathematics: #E4154B;
  ...
}
```

Listing 5.1: Design tokens for colors are implemented as CSS custom properties (global variables), which can then be used in various charts.

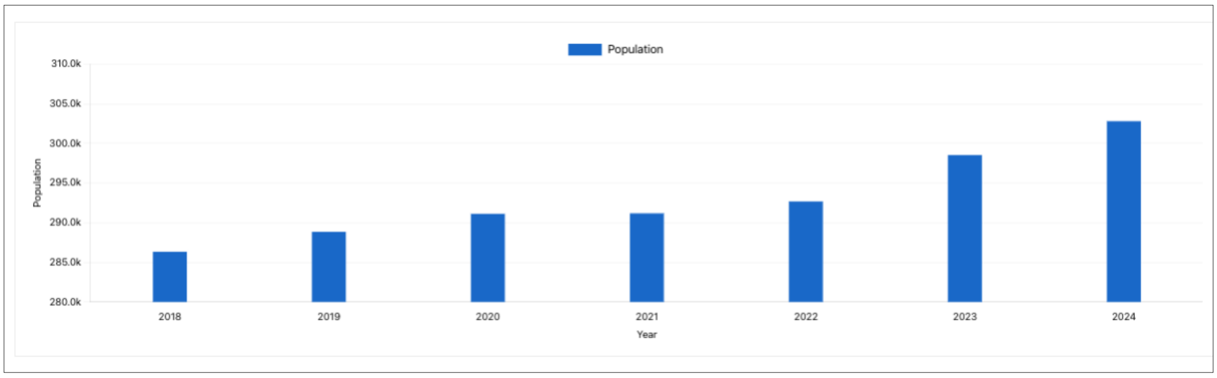


Figure 5.3: Graz population column chart. [Screenshot created by Laura Thaci.]

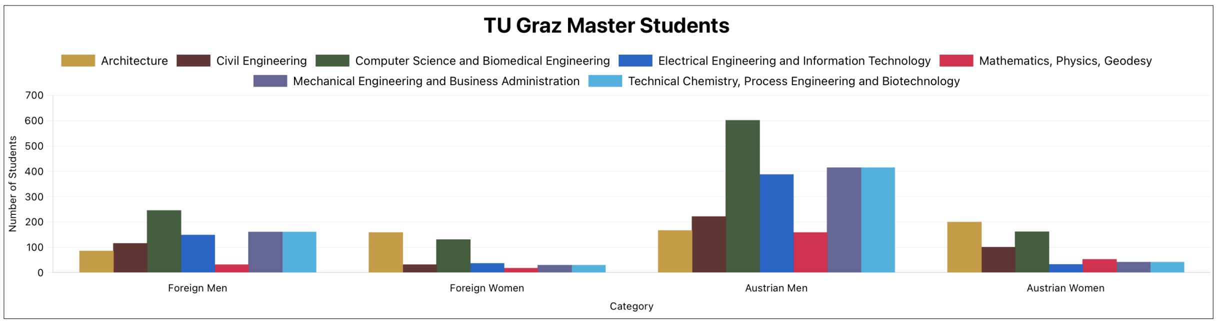


Figure 5.4: Bar chart of students per faculty. [Screenshot created by Laura Thaci.]

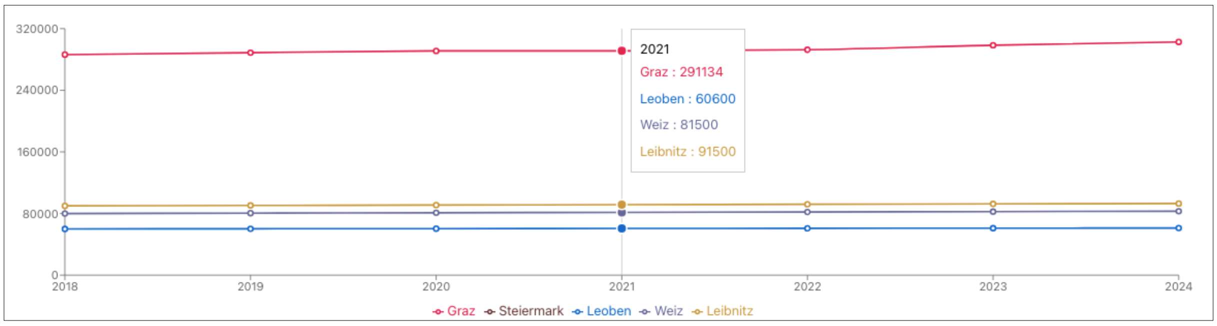
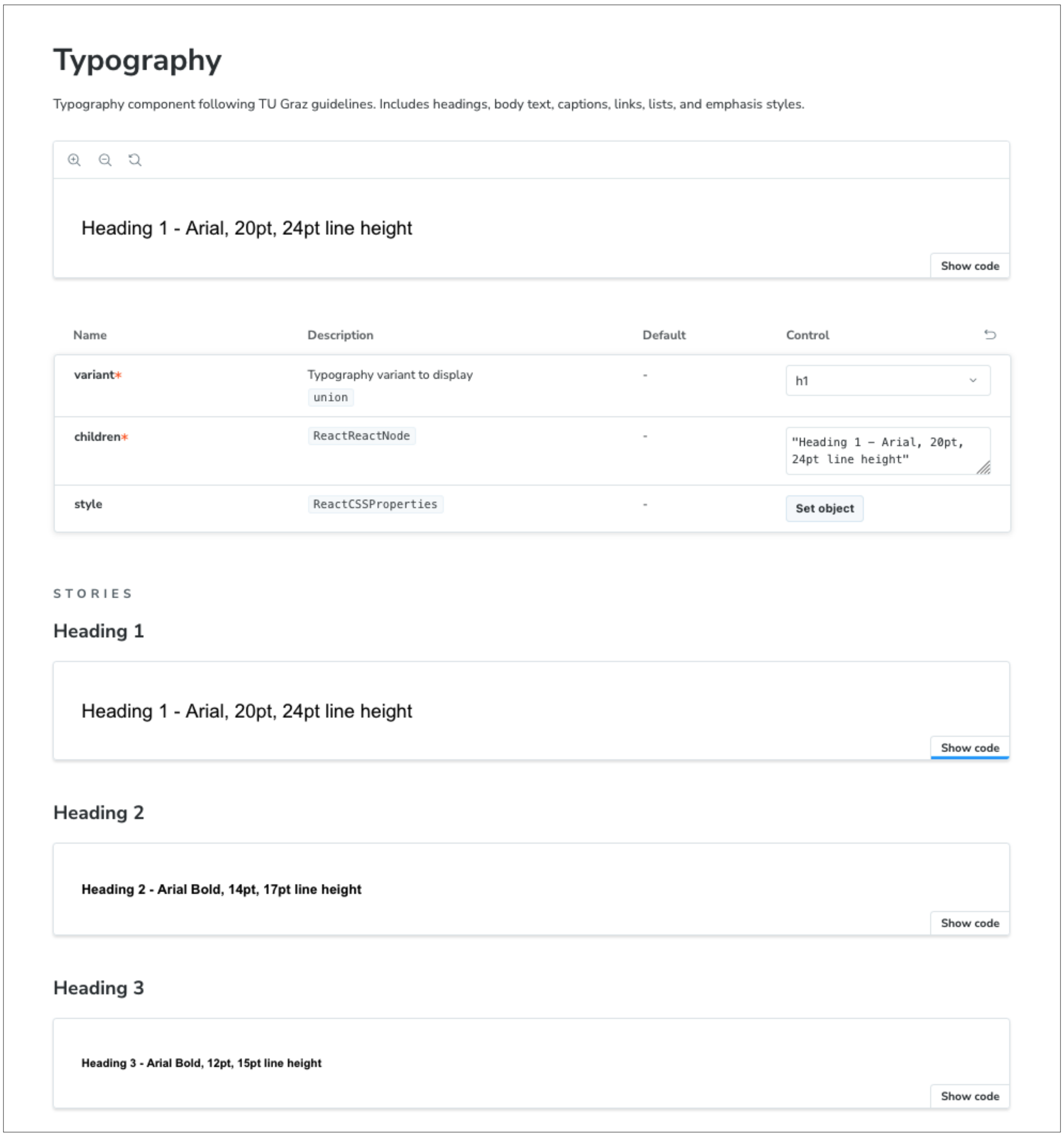


Figure 5.5: Graz population multi-line chart. [Screenshot created by Laura Thaci.]



Typography

Typography component following TU Graz guidelines. Includes headings, body text, captions, links, lists, and emphasis styles.

Heading 1 - Arial, 20pt, 24pt line height

Name	Description	Default	Control
variant*	Typography variant to display `union`	-	h1
children*	`ReactReactNode`	-	"Heading 1 - Arial, 20pt, 24pt line height"
style	`ReactCSSProperties`	-	Set object

STORIES

Heading 1

Heading 1 - Arial, 20pt, 24pt line height

Heading 2

Heading 2 - Arial Bold, 14pt, 17pt line height

Heading 3

Heading 3 - Arial Bold, 12pt, 15pt line height

Figure 5.6: Typography [Screenshot created by Laura Thaci.]

The `GrazPopulationMultiLineChart.tsx` component is built using Recharts. The component combines Recharts elements like `<LineChart>`, `<XAxis>`, `<YAxis>`, `<Tooltip>`, `<Legend>`, and several `<Line>` elements in a declarative way. The `data` prop (typically `populationTrends` with data for Graz, Leoben, Weiz, and Leibnitz) is provided to the `<LineChart>` component. Each `<Line>` component is then configured with a `dataKey` that matches one of these towns (e.g., “Graz”, “Leoben”). The component contains a fixed list that maps each town (key) to a specific TU Graz faculty color (as an example). This list is then used to create the `<Line>` elements with the appropriate settings.

`Typography.tsx` displays different Material-UI `<Typography>` components, each with different `variant` settings, to show the various text styles. This helps ensure text consistency across any application.

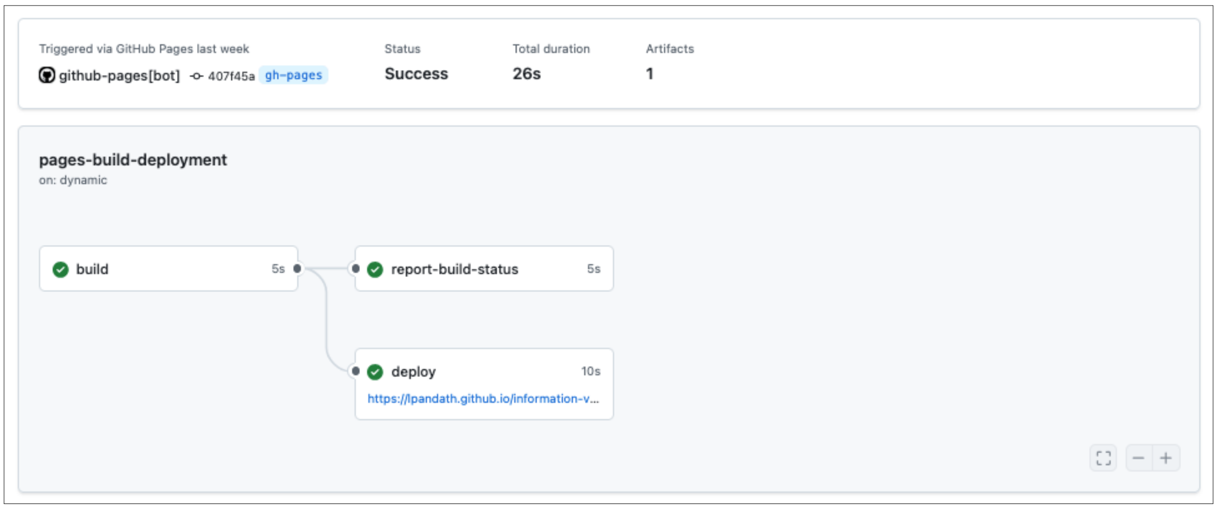

Triggered via GitHub Pages last week
github-pages[bot] -o- 407f45a gh-pages
Status
Success
Total duration
26s
Artifacts
1
pages-build-deployment
on: dynamic
build 5s
report-build-status 5s
deploy 10s
https://lpandath.github.io/information-v...

Figure 5.7: Storybook Deployment Workflow. [Screenshot created by Laura Thaci.]

5.3 Main Configuration

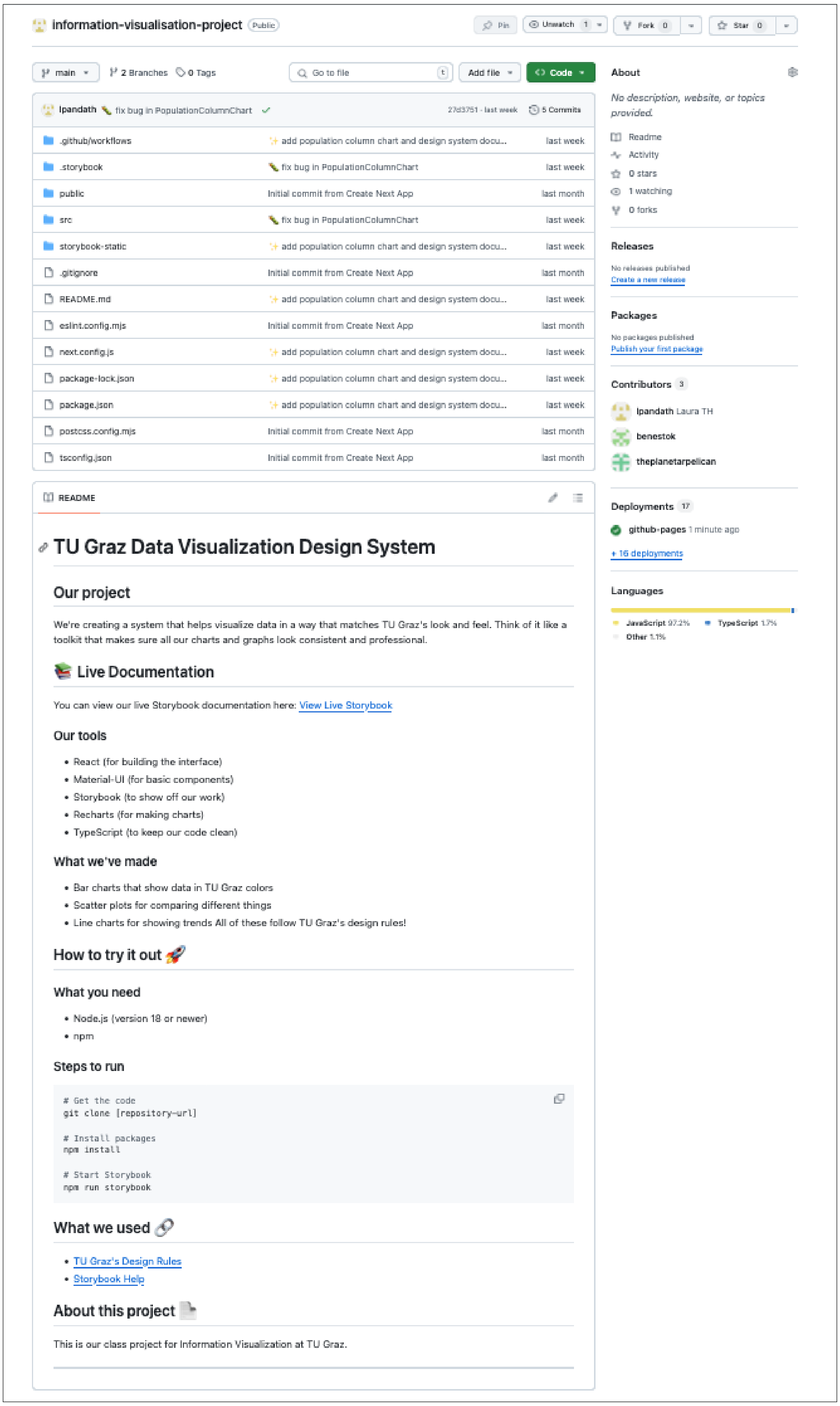
The `.storybook/main.ts` file contains global Storybook settings. It includes a listing of additional tools for Storybook (e.g., Docs, Controls, Actions, Viewport). The configuration also specifies the location of the `public/` folder, which contains static files like logos. The setup enables Storybook to work with React and TypeScript. The Storybook interface was customized to match TU Graz branding through theme settings.

5.4 GitHub Actions Workflow

A GitHub Actions workflow, implemented as an automated script in `.github/workflows/deploy.yml`, handles the deployment. The workflow first retrieves the latest code from the repository. It then configures the correct version of Node.js (e.g., v18) for the build. All required project packages are installed using `npm ci` for a clean build. The workflow executes `npm run build-storybook`, which generates the `storybook-static/` folder with all website files. Finally, the contents of `storybook-static/` are copied to the `gh-pages/` branch in the GitHub repository using the `JamesIves/github-pages-deploy-action@v4` tool.

5.5 GitHub Pages Configuration

In the GitHub repository settings, GitHub Pages was configured to display the website from the `gh-pages` branch. The live Storybook can then be seen at:
`https://lpandath.github.io/information-visualisation-project/`



information-visualisation-project Public

lpandath ✏️ fix bug in PopulationColumnChart ✓ 27d3751 · last week 5 Commits

.github/workflows	✨ add population column chart and design system docu...	last week
.storybook	✏️ fix bug in PopulationColumnChart	last week
public	Initial commit from Create Next App	last month
src	✏️ fix bug in PopulationColumnChart	last week
storybook-static	✨ add population column chart and design system docu...	last week
.gitignore	Initial commit from Create Next App	last month
README.md	✨ add population column chart and design system docu...	last week
eslint.config.mjs	Initial commit from Create Next App	last month
next.config.js	✨ add population column chart and design system docu...	last week
package-lock.json	✨ add population column chart and design system docu...	last week
package.json	✨ add population column chart and design system docu...	last week
postcss.config.mjs	Initial commit from Create Next App	last month
tsconfig.json	Initial commit from Create Next App	last month

About

No description, website, or topics provided.

Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 3

lpandath Laura TH
benestok
theplanetarpelican

Deployments 17

github-pages 1 minute ago
+ 16 deployments

Languages

JavaScript 97.2% TypeScript 1.7%
Other 1.1%

README

TU Graz Data Visualization Design System

Our project

We're creating a system that helps visualize data in a way that matches TU Graz's look and feel. Think of it like a toolkit that makes sure all our charts and graphs look consistent and professional.

📚 Live Documentation

You can view our live Storybook documentation here: View Live Storybook

Our tools

- React (for building the interface)
- Material-UI (for basic components)
- Storybook (to show off our work)
- Recharts (for making charts)
- TypeScript (to keep our code clean)

What we've made

- Bar charts that show data in TU Graz colors
- Scatter plots for comparing different things
- Line charts for showing trends All of these follow TU Graz's design rules!

How to try it out 🚀

What you need

- Node.js (version 18 or newer)
- npm

Steps to run

```
# Get the code
git clone [repository-url]

# Install packages
npm install

# Start Storybook
npm run storybook
```

What we used 🔗

- TU Graz's Design Rules
- Storybook Help

About this project 📄

This is our class project for Information Visualization at TU Graz.

Figure 5.8: Storybook Repository [Screenshot created by Laura Thaci.]

Chapter 6

Concluding Remarks

When embarking on a project to build a data visualization design system, a good approach to select a tool begins with clearly defining the intended message of the data that should be convey to the observer. Subsequently, one should identify a system, that includes chart examples similar to the envisioned output. The Adele design system collection offers useful filter options to find open-source solutions. It also enables users to search specifically for design systems that provide a Storybook interface, which is a great way to interact with standard chart components.

Among the presented examples, the Pajamas data visualization design system, discussed in Section 4.1.2, stands out, due to its open-source repository, detailed documentation, and a well-structured Storybook. An alternative for more complex data visualizations is the Carbon design system, discussed in Section 4.1.3, which offers the widest variety of different chart types. While the Dell Design System (see Section 4.2.1) also features a comprehensive Storybook, it cannot be freely used and is therefore recommended only as a source for inspiration for creating custom solutions.

The proposed prototype TU Graz Data Visualization Design System gives a good starting point for making charts that look consistent and use the TU Graz brand. The project put together React components, charting tools, and TU Graz design rules, with Storybook examples. The auto-deployment means the latest documentation is always online.

Bibliography

Andrews, Keith [2025]. *Information Visualisation: Lecture Notes*. 14 Mar 2025. https://courses.isds.tugraz.at/ivis/ivis.pdf (cited on page 6).

Callahan, Ben [2022]. *The Anatomy of a Design System*. 13 Apr 2022. https://sparkbox.com/foundry/design_system_makeup_design_system_layers_parts_of_a_design_system (cited on page 3).

Cesal, Amy [2019]. *What Are Data Visualization Style Guidelines?* 10 Jul 2019. https://nightingaledvs.com/what-are-data-visualization-style-guidelines/ (cited on page 9).

Chromatic [2025]. *Storybook*. https://storybook.js.org/ (cited on pages 1, 13, 19).

CodeSandbox [2025]. *CodeSandbox*. https://codesandbox.io/ (cited on page 13).

Czaban, Tom [2023]. *Top 10 Proven Data Visualization Best Practices*. Gooddata, 02 Nov 2023. https://gooddata.com/blog/5-data-visualization-best-practices/ (cited on page 11).

DataDog [2025]. *Druids*. 2025. https://druids.datadoghq.com/ (cited on page 16).

Dell [2025a]. *Dell Design System*. 2025. https://delldesignsystem.com/data-visualization/ddv-overview/ (cited on page 16).

Dell [2025b]. *Dell Design System: Button*. 12 May 2025. https://delldesignsystem.com/components/button/ (cited on pages 4–5).

Dell [2025c]. *Dell Storybook*. 2025. https://vanilla-dv.delldesignsystem.com/2.14.0/index.html?path=/docs/components-area-chart--basic (cited on page 16).

Dell [2025d]. *The Anatomy of a Stacked Area Chart*. 12 May 2025. https://delldesignsystem.com/data-visualization/area-chart/ (cited on page 10).

GitLab [2025a]. *Pajamas*. 11 May 2025. https://design.gitlab.com/ (cited on page 14).

GitLab [2025b]. *Pajamas Repository*. 11 May 2025. https://gitlab.com/gitlab-org/gitlab-services/design.gitlab.com (cited on page 14).

GitLab [2025c]. *Pajamas Storybook*. 11 May 2025. https://gitlab-org.gitlab.io/gitlab-ui/?path=%2Fstory%2Fcharts-column-chart--default (cited on page 14).

Google [2025]. *Angular*. The R Foundation. 2025. https://angular.dev/ (cited on page 15).

Greben, Ilya [2025a]. *Data Visualization*. Design Systems Database, 12 May 2025. https://designsystems.surf/directories/data-visualization (cited on page 9).

Greben, Ilya [2025b]. *Design Systems Surf*. https://designsystems.surf/design-systems (cited on page 13).

IBM [2025a]. *Carbon*. 11 May 2025. https://carbondesignsystem.com/data-visualization/getting-started/ (cited on page 15).

IBM [2025b]. *Carbon Repository*. 2025. https://github.com/carbon-design-system/carbon (cited on page 15).

IDF [2021]. *What are Design Systems?* Interaction Design Foundation, 05 Mar 2021. https://interaction-design.org/literature/topics/design-systems (cited on page 3).

Land Steiermark [2025]. *Population of Graz*. 2025. https://landesentwicklung.steiermark.at/cms/beitrag/12651292/141979459/ (cited on page 14).

MDN [2025]. *JavaScript — MDN Web Docs*. 11 May 2025. https://developer.mozilla.org/en-US/docs/Web/JavaScript (cited on page 13).

Meta [2025]. *React*. 2025. https://react.dev/ (cited on pages 13, 15, 19).

Michelin [2025]. *Motion*. 2025. https://designsystem.michelin.com/data-visualization/introduction (cited on page 16).

Microsoft [2025a]. *Blazor*. 2025. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor (cited on page 16).

Microsoft [2025b]. *TypeScript*. 2025. https://www.typescriptlang.org/ (cited on page 19).

Mir, Nabil [2023]. *Was sind Designsysteme?* 2023. https://99designs.de/blog/design-tipps/designsysteme/ (cited on page 6).

MUI [2025]. *React Material UI*. 2025. https://mui.com/ (cited on page 19).

Nearform [2025]. *Victory*. 2025. https://nearform.com/open-source/victory/ (cited on page 14).

Pessl, Laura [2025a]. *Carbon Showcase Video*. 2025. https://youtu.be/CSXDZgKc7dw (cited on page 16).

Pessl, Laura [2025b]. *Pajamas Showcase Video*. 11 May 2025. https://youtu.be/OFhINvk1kkA (cited on page 15).

Pessl, Laura [2025c]. *PatternFly Showcase Video*. 11 May 2025. https://youtu.be/B1neQtb4FQM (cited on page 14).

Ramamohan, Anirudh [2023]. *Design Systems Governance Framework*. 21 Mar 2023. https://medium.com/design-bootcamp/design-systems-governance-framework-799032d2f117 (cited on page 6).

Recharts [2025]. *Recharts*. 2025. https://recharts.org/ (cited on page 19).

Red Hat [2025a]. *PatternFly Open Source Design System*. 11 May 2025. https://github.com/patternfly/patternfly-org/tree/main (cited on page 13).

Red Hat [2025b]. *PatternFly*. https://patternfly.org/ (cited on page 13).

Svelte [2025]. *Svelte*. 2025. https://svelte.dev/ (cited on page 15).

Treder, Marcin [2025]. *Adele*. https://adele.uxpin.com/ (cited on page 13).

TU Graz [2019]. *TU Graz Corporate Design Richtlinie des Rektorates*. 10 Sep 2019. https://tugraz.at/fileadmin/user_upload/tugrazExternal/02bfe6da-df31-4c20-9e9f-819251ecfd4b/2019_2020/Stk_1/RL_92000_CODE_057_02_20190910.pdf (cited on page 19).

UXPin [2023]. *5 Key Design System Challenges and Lessons Learned*. 11 Apr 2023. https://uxpin.com/studio/blog/key-design-system-challenges/ (cited on page 7).

UXPin [2025]. *Adele*. https://github.com/UXPin/adele/blob/master/README (cited on page 13).

You, Evan [2025]. *Vue*. 2025. https://vuejs.org/ (cited on page 14).