EasyCharts: HTML Chart Elements with Web Components

Ville Määttä, Dorian Percic, Tomo Ratko

706.057 Information Visualisation SS 2022/2023
Graz University of Technology

03 July 2023

Abstract

This project work focuses on utilizing Web Components to create custom HTML elements, specifically line charts, multiline charts, and bar charts. By leveraging Web Components, the project aims to simplify the process of creating SVG charts by encapsulating the necessary logic within custom HTML elements. This approach enables end users to define and configure charts directly within the HTML markup, making it more intuitive and accessible.

The project is implemented using TypeScript, which allows for type safety and improved code organization. The npm package manager is utilized to manage dependencies and ensure a smooth development workflow. Gulp, a popular task runner, is chosen as the development tool to automate various build processes.

The D3.js library serves as the underlying engine for rendering the charts, providing a powerful and flexible set of features for data visualization. By utilizing D3.js in conjunction with Web Components, the project combines the strengths of both technologies to create a seamless and efficient charting solution.

Contents

Contents ii

List of Figures iii

List of Tables v

List of Listings vii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Custom Web Components . . . 1

1.3 Tools and Libraries . . . 2

2 Chart Properties 3

2.1 General Properties . . . 3

2.2 Internal Data Structure . . . 4

2.3 Data Series Mode . . . 4

2.4 Table Mode . . . 5

3 Line Chart 9

3.1 Code Realisation . . . 9

3.2 Examples . . . 10

4 Bar Chart 17

4.1 Code Realisation . . . 17

4.2 Examples . . . 17

5 Styling with CSS 23

5.1 Usage . . . 23

5.2 Chart Size . . . 23

5.3 Color . . . 23

5.4 Ticks Flag . . . 23

6 Limitations and Future Work 27

6.1 Limitations . . . 27

6.2 Future Work . . . 27

7 For Developers **29**
7.1 Installation 29
7.2 Project Structure 29
7.3 Build Process 29

Bibliography **33**

List of Figures

3.1 Line Chart . . . 10
3.2 Multiline Chart . . . 14

4.1 Bar Chart . . . 21

List of Tables

2.1 Chart Attributes . . . 4
2.2 Data Series Attributes . . . 5
2.3 Data Point Attributes . . . 6
2.4 <th> Attributes . . . 6
2.5 <td> Attributes . . . 7

5.1 CSS Attributes . . . 24

List of Listings

2.1 Internal Data Structure 4
2.2 Internal Data Structure 5
2.3 Single Data Series in Data Series Mode 6
2.4 Multiple Data Series in Data Series Mode 6
2.5 Single Data Series in Table Mode 7
2.6 Multiple Data Series in Table Mode 8

3.1 Line Chart Constructor and `connectedCallback` Method 11
3.2 `drawLineChart` Method, 1/3 12
3.3 `drawLineChart` Method, 2/3 13
3.4 `drawLineChart` Method, 3/3 14
3.5 Creating Line Chart in Data Series Mode 15
3.6 Creating Line Chart in Table Mode 16

4.1 BarChart Class Constructor and connectedCallback Function 18
4.2 drawBarChart Function Begin 19
4.3 Bar Chart Creation Using D3 20
4.4 drawBarChart Chart Axis Labelling 21
4.5 Bar Chart Usage 22

5.1 CSS Size Example 24
5.2 CSS Color Example 24
5.3 Ticks Flag 24
5.4 CSS Whole Example 25

7.1 Project Strcture 30

Chapter 1

Introduction

1.1 Motivation

Using charts in HTML is often a way harder task than it should be. Inserting a simple chart into a HTML requires the user to download and install bloated libraries, and implement their own chart using a programming language (e.g. JavaScript, TypeScript, etc.). These libraries often come with a lot of unnecessary functionalities for the end user, and some libraries even have to be bought.

What if that whole process could be avoided? What if someone could just include a file into their project and create CSS customizable charts using standard HTML? This idea brought EasyCharts project to life.

1.2 Custom Web Components

Web Components is a suite of different technologies making it possible to create reusable custom elements, with their functionality encapsulated away from the rest the code, and utilize them in web applications. [MDN contributors 2023a].

Web Components aim to solve problems of code messiness and reusability - it consists of three main technologies, which can be used together to create versatile custom elements with encapsulated functionality that can be reused wherever wanted, without fear of code collisions [MDN contributors 2023a]:

1. Custom elements:
    - A set of JavaScript APIs that allow one to define custom elements and their behavior, which can then be used as desired in the user interface.
2. Shadow DOM:
    - A set of JavaScript APIs for attaching an encapsulated "shadow" DOM tree to an element, which is rendered separately from the main document DOM - and controlling associated functionality. In this way, the element's features are kept private, so they can be scripted and styled without the fear of collision with other parts of the document.
3. HTML templates:
    - The `<template>` and `<slot>` elements enable one to write markup templates that are not displayed in the rendered page. These can then be reused multiple times as the basis of a custom element's structure.

The basic approach for implementing a web component generally looks something like this [MDN contributors 2023a]:

- Create a class in which the web component functionality is specified, using the `class` syntax.

- Register the new custom element using the `CustomElementRegistry.define()` method, passing it the element name to be defined, the class or function in which its functionality is specified, and optionally, what element it inherits from.
- If required, attach a shadow DOM to the custom element using `Element.attachShadow()` method. Add child elements, event listeners, etc., to the shadow DOM using regular DOM methods.
- If required, define an HTML template using `<template>` and `<slot>`. Again use regular DOM methods to clone the template and attach it to the shadow DOM. Use custom element wherever wanted, like for example on the web page, just like any regular HTML element would be used.

There are two types of custom elements [MDN contributors 2023b]:

- Autonomous custom elements: Standalone implementations, they don't inherit from standard HTML.
- Customized built-in elements: Components that inherit from standard HTML elements.

EasyCharts are implemented as autonomous custom web components.

1.3 Tools and Libraries

In EasyCharts, D3 was utilized, a JavaScript library, to create and manipulate Scalable Vector Graphics (SVG) chart elements. The project was written in TypeScript for better code organization, increased maintainability, and improved error detection. To manage project dependencies, Node.js (npm) was used as the package manager. Node.js enables the user to easily install and manage the various libraries and frameworks needed for the project.

Chapter 2

Chart Properties

There are several properties of the library, which hold for both the line chart as well as the bar chart. Together with some general properties, like x and y axis titles or error handling, the 2 data input mechanisms: Data series mode and table mode are discussed in this chapter in more detail.

2.1 General Properties

Using EasyCharts there are several properties, which hold for both chart types. One of them is the handling of custom errors. General description of the errors:

- When no table or data series is found, an error is thrown. This means either `<table>` or `<dataseries>` has to be defined, in order to parse the user inserted data points.
- Each dataseries element `<dataseries>` has to have a `name` attribute: `<dataseries name="dataseriesName ">`.
- At least one datapoint `<datapoint>` has to be inside a dataseries element.
- Each datapoint has to have a valid numeric value and it has to exist and each label has to exist.
- When in table mode, an invalid table structure error occurs, as soon as each row `<tr>` does not have at least two columns using `<td>` and both values in these columns have to exist and be valid.
- When in table mode at least on `<th>` has to be present in `<thead>`.

Another property includes the availability of CSS for styling the elements of the created chart, using HTML `name` and `id` attribute. More details for the CSS is discussed in Chapter 5. Adding x and y axis titles using `<x-axis-title>` or `<y-axis-title>` is also possible, as well as adding a custom chart title using the `chart-title` attribute in the root custom HTML element. Furthermore, the x and y axis are correctly scaled, based on the input data and the bars and lines of the respective chart are correctly labelled. Table of all properties for the charts is shown in Table 2.1.

There exist also 2 other HTML elements, which were not yet discussed into great detail:

- `<x-axis-title> X Axis Title </x-axis-title>`
- `<y-axis-title> Y Axis Title </y-axis-title>`

With the `<x-axis-title>` element, it is possible to set the x axis title, whereas with `<y-axis-title>` it is possible to set the y axis title.

Attribute	Description
height	Height of the chart.
width	Width of the chart.
id	Unique id for the chart. Used for setting the height and width of the chart, or specifying if the x axis ticks should be shown, through CSS.
class	Class attribute of the chart. Used for setting the height and width of the chart, or specifying if the x axis ticks should be shown, through CSS.
chart-title	Title of the chart.

Table 2.1: All possible chart attributes. Holds for all chart types.

```
{
  "dataseries1": {
    "1960": 3.03,
    "1970": 3.70,
    "1980": 4.43,
    "1990": 5.28,
    "2000": 6.08,
    "2010": 6.87
  }
}
```

Listing 2.1: Internal data structure for displaying charts.

2.2 Internal Data Structure

When visualizing information and handling custom user input a good data structure is the key to success of, on the one side, clean and concise code and on the other side of an efficient processing capability. Hence some thought should be invested, when constructing the data structure.

After processing the data gotten from the HTML table or the custom defined data series, the same data structure is constructed. It is best to instantly look on an example, defined in Listing 2.3. In this example the internal data structure would look like in Listing 2.1. This means it basically has a dictionary of dictionaries. The first dictionary with key `"dataseries1"` is, as one can see, the name of the data series. All other elements in this dictionary are again in a dictionary and the key for them is the label name and the value the numeric data point value. When looking at Listing 2.2, whereas the data is from Listing 2.4, one can see the data structure created from the data series. It is important to see, that the name of the data series is the key again of the outer directory and in multi line chart this is then the line name basically, so the name, which stands besides the line when displaying the chart. When using single series charts, the name attribute is only important to achieve this data structure.

2.3 Data Series Mode

In the data series mode the user is able to input data using using a custom defined input mechanism called `<dataseries>` and `<datapoint>`. The data series element defines, as the name suggests, a data series and therefore one can create multi series charts. In case of EasyCharts, of course only multiline chart is possible. Each data series has to have a name, as this lies in the nature of the internal data structure. Each

```
{
  "SalesTeam": {
    "6AM": 5,
    "12PM": 10,
    "6PM": 15,
    "12AM": 1
  },
  "SumOfSales": {
    "6AM": 10,
    "12PM": 15,
    "6PM": 20,
    "12AM": 5
  }
}
```

Listing 2.2: Internal data structure for displaying charts.

Attribute	**Description**
name	Name of the data series. Necessary for the internal data structure. In multiline chart it represents the line name.
id	Unique id for data series. Used for CSS in linechart for the line color.
class	Class attribute of the data series. Used for CSS in linechart for the line color.

Table 2.2: All possible data series attributes.

data point has 2 values, separated by a comma (`,`). In this example: `<datapoint>a, 1</datapoint>`, the label, which is displayed on the x axis, will be "`a`" and the corresponding value will be "`1`". The attributes of `<dataseries>` can be seen in Table 2.2 and of `<datapoint>` in Table 2.3.

In Listing 2.3 an example of a single series line chart using data series can be seen. In Listing 2.4 an example of a multiline chart can be seen with multiple data series.

2.4 Table Mode

In table mode, the user can specify data in standard HTML format. In this way, the user can think of data as it is usual with Comma-Separated Values (CSV) files, probably making it easier for them to construct the charts. The table has to have a `<thead>` element as well as a `<tbody>` element. Inside `<thead>` there has to exist one `<tr>` element and inside this row at least one `<th>` element has to exist, else an error is thrown. The `<th>` element represents the data series name. This replaces the "`name`" attribute in data series mode. Inside `<tbody>` one can define multiple rows - defined with `<tr>` - which defines the amount of x axis values. Inside a `<tr>` there has to be at least one `<td>` element. The first `<td>` element is always the label for the point. Every other `<td>` element is a data point and if for example there are three `<td>` elements, then it implies there are two data series. The data structure of course looks exactly the same as in Listing 2.2 or Listing 2.1, if only single series. The attributes of the `<th>` element can be seen in Table 2.4 and the attributes of `<td>` can be seen in Table 2.5.

In Listing 2.5 an example of a single series line chart using a HTML table can be seen. In Listing 2.6 an example of a multiline chart can be seen with multiple data series in HTML table mode.

Year	World Population (in billions)
id	Unique id for data point. Used for CSS in bar chart for the bar color.
class	Class attribute of the data point. Used for CSS in bar chart for the bar color.

Table 2.3: All possible data point attributes.

```
<ec-linechart>
  <dataseries name="dataseries1">
    <datapoint> 1960, 3.03 </datapoint>
    <datapoint> 1970, 3.70 </datapoint>
    <datapoint> 1980, 4.43 </datapoint>
    <datapoint> 1990, 5.28 </datapoint>
    <datapoint> 2000, 6.08 </datapoint>
    <datapoint> 2010, 6.87 </datapoint>
  </dataseries>
</ec-linechart>
```

Listing 2.3: Single data series using the data series mode.

```
<ec-linechart id="mlinechart1">
  <x-axis-title>Time</x-axis-title>
  <y-axis-title>Profit in Thousands(€)</y-axis-title>
  <dataseries name="SalesTeam"
  id="samsung">
    <datapoint> 6AM,5 </datapoint>
    <datapoint> 12PM,10 </datapoint>
    <datapoint> 6PM,15 </datapoint>
    <datapoint> 12AM,1 </datapoint>
  </dataseries>
  <dataseries name="SumOfSales"
  id="apple">
    <datapoint> 6AM,10 </datapoint>
    <datapoint> 12PM,15 </datapoint>
    <datapoint> 6PM,20 </datapoint>
    <datapoint> 12AM,5 </datapoint>
  </dataseries>
</ec-linechart>
```

Listing 2.4: Multiple data series using the data series mode.

Attribute	Description
id	Unique id for data series represented in `<th>`. Used for CSS in line chart for the line color.
class	Class attribute of data series in `<th>`. Used for CSS in line chart for the line color.

Table 2.4: All possible `<th>` attributes.

Attribute	Description
id	Unique id for data point represented in `<td>`. Used for CSS in bar chart for the bar color.
class	Class attribute of data point in `<td>`. Used for CSS in bar chart for the bar color.

Table 2.5: All possible `<td>` attributes.

```
<ec-barchart class="barchart"
  chart-title="Styrian Election Results in 2015">
  <x-axis-title> Party </x-axis-title>
  <y-axis-title> Percent </y-axis-title>
  <table>
    <thead>
      <tr>
        <th>Party Results</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td id="spoe">SPÖ</td>
        <td>29.29</td>
      </tr>
      <tr>
        <td id="oevp">ÖVP</td>
        <td>28.45</td>
      </tr>
      <tr>
        <td id="fpoe">FPÖ</td>
        <td>26.76</td>
      </tr>
      <tr>
        <td id="gruene">Grüne</td>
        <td>6.68</td>
      </tr>
      <tr>
        <td id="kpoe">KPÖ</td>
        <td>4.22</td>
      </tr>
      <tr>
        <td id="neos">Neos</td>
        <td>2.64</td>
      </tr>
      <tr>
        <td id="frank">Frank</td>
        <td>1.74</td>
      </tr>
    </tbody>
  </table>
</ec-barchart>
```

Listing 2.5: Single data series using the table mode.

```
<ec-linechart id="mlinechart1" chart-title="Profits: Apple vs Samsung">
  <x-axis-title>Time</x-axis-title>
  <y-axis-title>Profit in Thousands(€)</y-axis-title>
    <table>
      <thead>
        <tr>
          <th id="samsung">Samsung</th>
          <th id="apple">Apple</th>
        </tr>
    </thead>
      <tbody>
        <tr>
          <td>6AM</td>
          <td>5</td>
          <td>10</td>
        </tr>
        <tr>
          <td>12PM</td>
          <td>10</td>
          <td>15</td>
        </tr>
        <tr>
          <td>6PM</td>
          <td>15</td>
          <td>20</td>
        </tr>
        <tr>
          <td>12AM</td>
          <td>1</td>
          <td>5</td>
        </tr>
    </tbody>
  </table>
</ec-linechart>
```

Listing 2.6: Multiple data series using the table mode.

Chapter 3

Line Chart

Line Chart is the second and last chart type so far, which was implemented. It works both with table mode and with data series mode. Furthermore, not only a single series line chart is implemented, but also multi series line chart (multiline chart)., which allows more complex visualizing possibilities.

3.1 Code Realisation

The line chart is implemented in similar fashion as the bar chart i.e. as a custom `LineChart` class and using D3 library for drawing the actual SVG charts. As discussed in the Chapter 2, line charts are possible to create in HTML using either table mode or data series mode. Multiline charts are also possible to create, just by inserting multiple data series.

The `LineChart` class has several methods and utility functions that are used to draw and handle the line chart. Here is an overview of the code:

1. The constructor method is called when an instance of the `LineChart` class is created. It initializes the class by calling the `super()` method to invoke the constructor of the parent class (HTMLElement), and then it attaches a Shadow DOM to the element.

2. The `connectedCallback()` method is a lifecycle method that is called when the web component is attached to the DOM. It performs the following tasks:

   - It determines the width and height of the chart by calling the `setChartWidth()` and `setChartHeight()` utility functions.

   - It checks the structure of the chart by looking for a <`table`> or <`dataseries`> element within the web component.

   - If a <`table`> element is found, it calls the `handleTableMode()` function to handle the chart drawing logic for table-based data.

   - If a <`dataseries`> element is found, it calls the `handleDataSeriesMode()` function to handle the chart drawing logic for data series-based data.

   - If neither a <`table`> nor a <`dataseries`> element is found, it logs an error message.

3. The `drawLineChart()` method is responsible for actually drawing the line chart and adding it to the Shadow DOM. It takes the width, height, and data dictionaries as parameters. The method uses D3.js library functions to generate and manipulate SVG elements to create the line chart based on the provided data.

4. After defining the `LineChart` class, the code uses `customElements.define()` to register the custom element with the name `ec-linechart`, allowing it to be used in HTML.

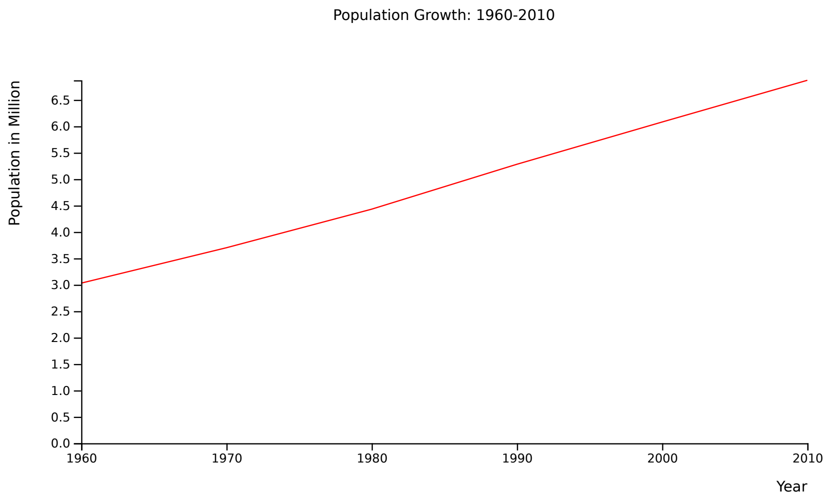

Figure 3.1: Line chart showing the population growth from 1960 to 2010.

5. Following the class definition, there are several utility functions used by the `LineChart` class, but also by the `BarChart` class:
   - `getTableDict()` extracts data from a `<table>` element and returns a dictionary of data points and axis headers.
   - `getDataSeriesDict()` extracts data from `<dataseries>` elements and returns a dictionary of data points and axis headers.
   - `getDictValues()` returns an array of values from a given dictionary.
   - `setChartWidth()` and `setChartHeight()` extract the chart width and height from the HTML attributes of the web component.
   - `isValidNumber()` checks if a given string is a valid number.

Overall, the code provides a foundation for creating and displaying line charts as custom web components using D3.js and HTML elements like `<table>` and `<dataseries>`. The `LineChart` class encapsulates the chart functionality and can be extended or customized further as needed.

3.2 Examples

Example of creating line chart using data series mode can be seen in Listing 3.5 and the actual resulting line chart in Figure 3.1. Example of creating multiline chart using table mode can be seen in Listing 3.6 and the result in Figure 3.2. An example usage of the line chart can be seen in this video: `https://youtu.be/UXe0OD8OIf8` and of the multiline chart in this video: `https://youtu.be/A2Egg41JstY`.

```
class LineChart extends HTMLElement {
  /**
   * Constructor of web component, create Shadow DOM.
  */
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  /**
   * This life-cycle method will be called as soon as the web
   * component is attached to the DOM.
  */
  connectedCallback() {
    // Set size, CSS has most precedence
    let width = setChartWidth(this);
    let height = setChartHeight(this);
    const attributes = getSizeFromCSS(this);
    const showTicks = getTicksFlagFromCSS(this);
    if (attributes) {
      if (attributes['height']) {
        height = attributes['height'];
      }
      if (attributes['width']) {
        width = attributes['width'];
      }
    }
    // Set chart title
    let chartTitle = this.getAttribute('chart-title');
    if (!chartTitle) {
      chartTitle = 'chartTitle';
    }

    if (this.querySelector('table')) {
      handleTableMode(this, width, height, showTicks, chartTitle);
    } else if (this.querySelector('dataseries')) {
      handleDataSeriesMode(this, width, height, showTicks, chartTitle);
    } else {
      console.error(
        '[Error] Invalid chart structure: No table or data series found'
      );
    }
  }
  [...]
```

Listing 3.1: LineChart class constructor and `connectedCallback` function.

```
[...]
drawLineChart(
  width: number,
  height: number,
  dictionaries: [
    { [key: string]: { [innerKey: string]: number } },
    { [key: string]: string },
    { [key: string]: string }
  ],
  showTicks: string = 'true',
  chartTitle: string = 'chartTitle'
): void {
  const margin = {
    top: height * 0.2,
    bottom: height * 0.2,
    left: width * 0.2,
    right: width * 0.2,
  };
  const data = dictionaries[0];
  const axisTitles = dictionaries[1];
  const colors = dictionaries[2];

  const xScale = scalePoint()
    .domain(
      Object.keys(data).reduce((labels, seriesKey) => {
        const seriesLabels = Object.keys(data[seriesKey]);
        return [...labels, ...seriesLabels];
      }, [])
    ).range([margin.left, width - margin.right]);

  const yScale = scaleLinear()
    .domain([
      0,
      max(Object.values(data).map((line) => max(Object.values(line)))),
    ]).range([height - margin.bottom, margin.top]);

  const lineChartSvg = create('svg')
    .attr('viewBox', `0 0 ${width} ${height}`)
    .attr('version', '1.1');
}
[...]
```

Listing 3.2: Scaling and some variable definitions in `drawLineChart` method. `drawLineChart` code snippet 1/3.

```
[...]
// Draw lines
const lineGenerator = line<{ x: string; y: number }>()
  .x((d) => xScale(d.x) || 0)
  .y((d) => yScale(d.y) || 0);
Object.entries(data).forEach(([key, lineData]) => {
  lineChartSvg
    .append('path')
    .datum(Object.entries(lineData).map(([x, y]) => ({ x, y })))
    .attr('d', lineGenerator)
    .attr('fill', 'none')
    .attr('stroke', colors[key])
    .attr('id', key);

  if (Object.entries(data).length > 1) {
    // Get the last point of the line
    const lastPoint = Object.entries(lineData).slice(-1)[0];
    const [lastX, lastY] = lastPoint;

    // Display key beside the end of the line
    lineChartSvg
      .append('text')
      .attr('class', 'line-key')
      .attr('x', xScale(lastX) + 10)
      .attr('y', yScale(lastY) + 5)
      .attr('fill', colors[key])
      .text(key);
    }
});
const xAxis =
  showTicks === 'false'
    ? axisBottom(xScale).tickValues([])
    : axisBottom(xScale);

lineChartSvg
  .append('text')
  .attr('id', 'chart-title')
  .attr('text-anchor', 'middle')
  .attr('x', width / 2)
  .attr('y', margin.top / 2)
  .text(chartTitle);

lineChartSvg
  .append('g')
  .attr('transform', `translate(0, ${height - margin.bottom})`)
  .attr('id', 'x-axis')
  .call(xAxis);

lineChartSvg
  .append('g')
  .attr('transform', `translate(${margin.left}, 0)`)
  .attr('id', 'y-axis')
  .call(axisLeft(yScale));
[...]
```

Listing 3.3: Creating SVG element and drawing the lines in `drawLineChart` method. `drawLineChart` code snippet 2/3.

```
[...]
// X labels
lineChartSvg
  .append('text')
  .attr('id', 'x-label')
  .attr('text-anchor', 'end')
  .attr('x', width - margin.right)
  .attr('y', height - margin.bottom + 40)
  .text(axisTitles['x-axis']);

// Y labels
lineChartSvg
  .append('text')
  .attr('id', 'y-label')
  .attr('text-anchor', 'end')
  .attr('x', -margin.bottom)
  .attr('y', margin.left - 60)
  .attr('dy', '.75em')
  .attr('transform', 'rotate(-90)')
  .text(axisTitles['y-axis']);

this.shadowRoot?.appendChild(lineChartSvg.node());
[...]
```

Listing 3.4: Adding labels to the chart in `drawLineChart` method. `drawLineChart` code snippet 3/3.

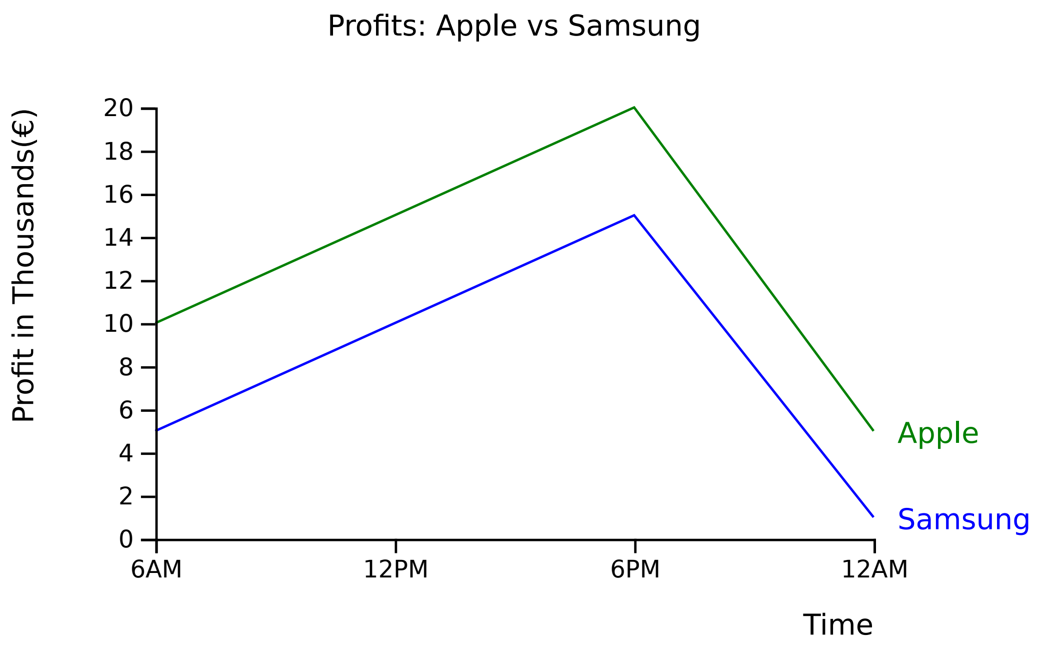

Figure 3.2: Multiline chart showing profits of Apple vs Samsung in one day.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Line Chart</title>
    <script src="./easycharts.js" defer></script>
    <style>
      .linechart {
        --chart-width: 1000;
        --chart-height: 500;
        --show-ticks: true;
      }
      #dataseries1 {
        --color: red;
      }
    </style>
    <!--External stylesheet only possible if used with server (npx gulp)!-->
    <!-- <link rel="stylesheet" href="./style.css"> -->
  </head>

  <body>
    <h1>Line Chart Example[Dataseries]</h1>
    <ec-linechart class="linechart" chart-title="Population Growth: 1960-2010">
      <x-axis-title>Year</x-axis-title>
      <y-axis-title>Population in Million</y-axis-title>
      <dataseries name="dataseries1" id="dataseries1">
        <datapoint> 1960,3.03 </datapoint>
        <datapoint> 1970,3.7 </datapoint>
        <datapoint> 1980,4.43 </datapoint>
        <datapoint> 1990,5.28 </datapoint>
        <datapoint> 2000,6.08 </datapoint>
        <datapoint> 2010,6.87 </datapoint>
      </dataseries>
    </ec-linechart>
  </body>
</html>
```

Listing 3.5: Creating a line chart in HTML using data series mode.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Multiline Chart</title>
    <script src="./easycharts.js" defer></script>
    <style>
      #mlinechart1 {
        --chart-width: 500;
        --chart-height: 300;
      }
      #samsung {
        --color: blue;
      }
      #apple {
        --color: green;
      }
    </style>
    <!--External stylesheet only possible if used with server (npx gulp)!-->
    <!-- <link rel="stylesheet" href="./style.css"> -->
  </head>

  <body>
    <h1>Multiline Chart Example[Table]</h1>
    <ec-linechart id="mlinechart1" chart-title="Profits: Apple vs Samsung">
      <x-axis-title>Time</x-axis-title>
      <y-axis-title>Profit in Thousands(€)</y-axis-title>
      <table>
        <thead>
          <tr>
            <th id="samsung">Samsung</th>
            <th id="apple">Apple</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>6AM</td>
            <td>5</td>
            <td>10</td>
          </tr>
          <tr>
            <td>12PM</td>
            <td>10</td>
            <td>15</td>
          </tr>
          <tr>
            <td>6PM</td>
            <td>15</td>
            <td>20</td>
          </tr>
          <tr>
            <td>12AM</td>
            <td>1</td>
            <td>5</td>
          </tr>
        </tbody>
      </table>
    </ec-linechart>
  </body>
</html>
```

Listing 3.6: Creating a multi-line chart in HTML using table mode.

Chapter 4

Bar Chart

The code is very similar to the one of the line chart, only the `drawLineChart` and `drawBarChart` are different, hence for other functions please refer to Chapter 3. However, a short explanation is still provided.

4.1 Code Realisation

The bar chart was implemented as a custom class as seen in Listing 4.1 and then appended to the controller of custom elements on a web document, the `CustomElementRegistry` object. This object allows registering a custom element on the page, return information on what custom elements are registered, etc.

Inside the method `connectedCallback`, all the functionality of the element is defined, as soon as it is connected to the DOM.

The chart is created as an SVG element using the D3 library, as seen in Listing 4.3 and Listing 4.4.

4.2 Examples

An example of using an EasyCharts bar chart with table mode can be seen in Listing 4.5 and the final result can be seen in Figure 4.1. In this video: `https://youtu.be/JKVT2ABBfz0` an example of a bar chart can be seen.

```
class BarChart extends HTMLElement {
  /**
   * Constructor of web component, create Shadow DOM.
   */
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  /**
   * This life-cycle method will be called as soon as the web component is attached
      to the DOM.
   */
  connectedCallback() {
    // Set size, CSS has most precedence
    let width = setChartWidth(this);
    let height = setChartHeight(this);
    const attributes = getSizeFromCSS(this);
    const showTicks = getTicksFlagFromCSS(this);
    if (attributes) {
      if (attributes['height']) {
        height = attributes['height'];
      }
      if (attributes['width']) {
        width = attributes['width'];
      }
    }
    // Set chart title
    let chartTitle = this.getAttribute('chart-title');
    if (!chartTitle) {
      chartTitle = 'chartTitle';
    }

    if (this.querySelector('table')) {
      handleTableMode(this, width, height, showTicks, chartTitle);
    } else if (this.querySelector('dataseries')) {
      handleDataSeriesMode(this, width, height, showTicks, chartTitle);
    } else {
      console.error(
        '[Error] Invalid chart structure: No table or data series found'
      );
    }
  }
  [...]
```

Listing 4.1: BarChart class constructor and connectedCallback function.

```
[...]
/**
 * Function drawing the chart and adding it to the Shadow DOM.
 * @param width - Width of the chart.
 * @param height - Height of the chart.
 * @param dictionaries - Array of 3 dictionaries: First consisting of the
    datapoints,
 * second of the x and y axis titles and the third of the colors.
 * @param showTicks - Flag indicating whether to show ticks on the x-axis. Default
     is 'true'.
 * @param chartTitle - Title of the chart.
 */
drawBarChart(
  width: number,
  height: number,
  dictionaries: [
    { [key: string]: { [innerKey: string]: number } },
    { [key: string]: string },
    { [key: string]: string }
  ],
  showTicks: string = 'true',
  chartTitle: string = 'chartTitle'
): void {
  const margin = {
    top: height * 0.2,
    bottom: height * 0.2,
    left: width * 0.2,
    right: width * 0.2,
  };
  const dataDict = dictionaries[0];
  const axisTitles = dictionaries[1];
  const colors = dictionaries[2];
  const dataSeriesKey = Object.keys(dataDict)[0];
  // Only one dataseries possible for bar chart
  const dataArray = getDictValues(dataDict[dataSeriesKey]);

  const chartWidth = width - margin.left - margin.right;
  const chartHeight = height - margin.top - margin.bottom;
  [...]
```

Listing 4.2: Begin of drawBarChart function.

```
[...]
const barChartSvg = create('svg')
  .attr('viewBox', `0 0 ${width} ${height}`)
  .attr('version', '1.1');

const xScale = scaleBand().range([0, chartWidth]).padding(0.1);

xScale.domain(Object.keys(dataDict[dataSeriesKey]));

const yScale = scaleLinear()
  .domain([0, max(dataArray) || 0])
  .range([chartHeight, 0]);

const xAxis =
  showTicks === 'false'
    ? axisBottom(xScale).tickValues([])
    : axisBottom(xScale);
const yAxis = axisLeft(yScale).ticks(5);

barChartSvg
  .append('text')
  .attr('id', 'chart-title')
  .attr('text-anchor', 'middle')
  .attr('x', width / 2)
  .attr('y', margin.top / 2)
  .text(chartTitle);

barChartSvg
  .append('g')
  .attr('transform', `translate(${margin.left}, ${height - margin.bottom})`)
  .attr('id', 'x-axis')
  .call(xAxis);

barChartSvg
  .append('g')
  .attr('transform', `translate(${margin.left}, ${margin.top})`)
  .attr('id', 'y-axis')
  .call(yAxis);

// Append bars
barChartSvg
  .append('g')
  .selectAll('rect')
  .data(dataArray)
  .join('rect')
  .attr(
    'x', (_, i) => margin.left + xScale(Object.keys(dataDict[dataSeriesKey])[i])
  )
  .attr('id', (_, i) => Object.keys(dataDict[dataSeriesKey])[i])
  .attr('y', (d) => margin.top + yScale(d))
  .attr('width', xScale.bandwidth())
  .attr('height', (d) => chartHeight - yScale(d))
  .attr('fill', (_, i) => colors[Object.keys(dataDict[dataSeriesKey])[i]]);
  [...]
```

Listing 4.3: Bar chart creation using D3 library functions.

```
    [...]
    // X labels
    barChartSvg
      .append('text')
      .attr('id', 'x-label')
      .attr('text-anchor', 'end')
      .attr('x', width - margin.right)
      .attr('y', height - margin.bottom + 40)
      .text(axisTitles['x-axis']);

    // Y labels
    barChartSvg
      .append('text')
      .attr('id', 'y-label')
      .attr('text-anchor', 'end')
      .attr('x', -margin.bottom)
      .attr('y', margin.left - 60)
      .attr('dy', '.75em')
      .attr('transform', 'rotate(-90)')
      .text(axisTitles['y-axis']);

    this.shadowRoot?.appendChild(barChartSvg.node());
  }
}
```

Listing 4.4: Snippet of axis labelling for bar chart.

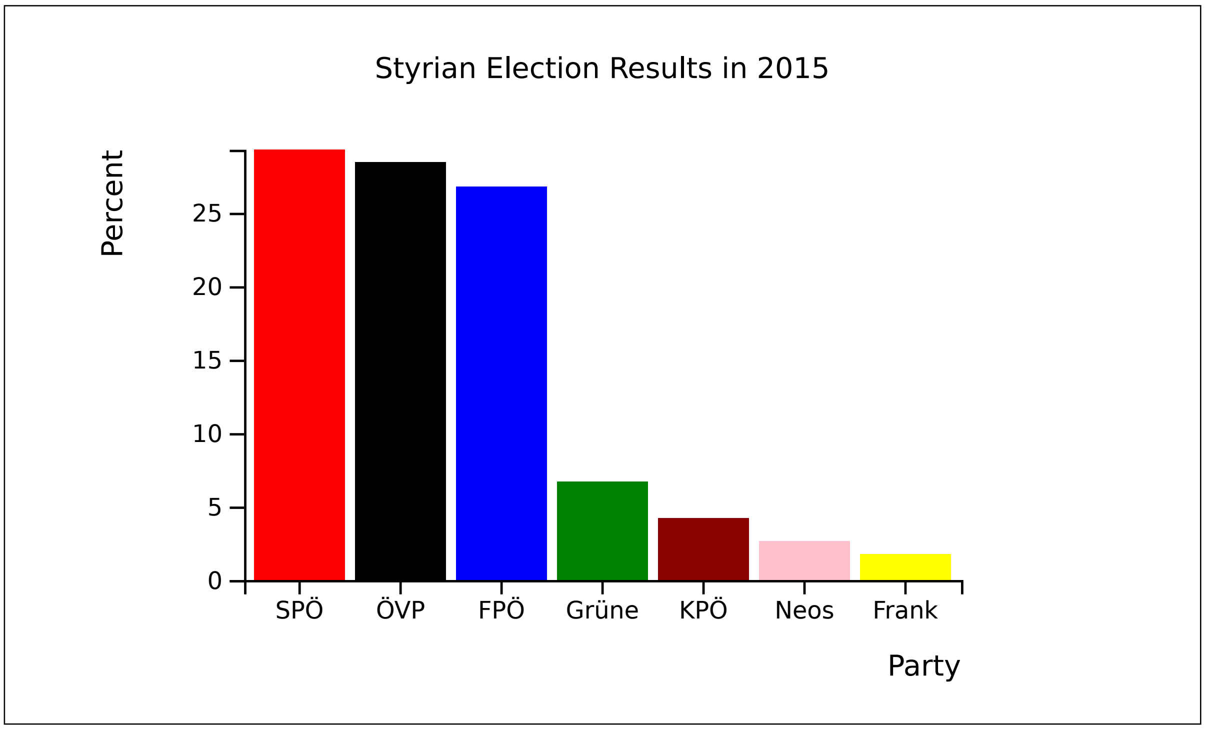

Figure 4.1: Bar chart showing the Styrian election results in 2015.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Bar Chart</title>
    <script src="./easycharts.js" defer></script>
    <!--External stylesheet only possible if used with server (npx gulp)!-->
    <!-- <link rel="stylesheet" href="./style.css"> -->
    <style>
      ...
    </style>
  </head>

  <body>
    <h1>Bar Chart Example[Table]</h1>
    <ec-barchart
      class="barchart"
      chart-title="Styrian Election Results in 2015">
      <x-axis-title> Party </x-axis-title>
      <y-axis-title> Percent </y-axis-title>
      <table>
        <thead>
          <tr>
            <th>Party Results</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td id="spoe">SPÖ</td>
            <td>29.29</td>
          </tr>
          <tr>
            <td id="oevp">ÖVP</td>
            <td>28.45</td>
          </tr>
          <tr>
            <td id="fpoe">FPÖ</td>
            <td>26.76</td>
          </tr>
          <tr>
            <td id="gruene">Grüne</td>
            <td>6.68</td>
          </tr>
          <tr>
            <td id="kpoe">KPÖ</td>
            <td>4.22</td>
          </tr>
          <tr>
            <td id="neos">Neos</td>
            <td>2.64</td>
          </tr>
          <tr>
            <td id="frank">Frank</td>
            <td>1.74</td>
          </tr>
        </tbody>
      </table>
    </ec-barchart>
  </body>
</html>
```

Listing 4.5: Bar chart created with HTML table.

Chapter 5

Styling with CSS

Styling with CSS is an important aspect of a good user experience when doing work with HTML. Therefore it is possible to style some elements of the created charts using standard CSS syntax. General usage of the defined CSS as well as all possibilities on what to style on the charts are analyzed in the following chapters. In each chapter one small example is shown, however in Listing 5.4 a more detailed example is provided.

5.1 Usage

In general, there are 2 ways on how to style components with CSS:

1. Through id's: Usage with `#id` in CSS.
2. Through class attributes: Usage with `.class` in CSS.

To see which CSS property belongs to which element look for the tables in the previous chapters. Table showing all currently available CSS properties can be seen in Table 5.1. In order to link external CSS files, one has to run a server. Without a server only with an inline `<style>` element the styling of the components will work.

5.2 Chart Size

Setting the chart size works with `--chart-width` or `--chart-height`. The size is defined as an integer and this integer is then the input for the SVG viewbox. An example of setting the size can be shown in Listing 5.1.

5.3 Color

Setting the chart color works with the `--color` CSS element. Using that property, one can set the color of the line or bar, defining a custom color. An example of changing the color can be shown in Listing 5.2.

5.4 Ticks Flag

Specifying if the x axis ticks should be shown is also possible with CSS. If the `--show-ticks` flag is set to true, then it will be shown. However, if the `--show-ticks` flag is set to false, then the ticks won't be shown. Example usage can be seen in Listing5.3.

Property	Description
–color	Set color of data series/data point.
–chart-height	Set height of the chart.
–chart-width	Set width of the chart.
–show-ticks	Flag either "true" or "false" for specifying if x axis ticks should be shown. Default is "true".

Table 5.1: All possible CSS attributes.

```
#barchart1 {
--chart-width: 1000;
--chart-height: 300;
}
[...]
<ec-barchart id="barchart1">
```

Listing 5.1: Set the size with CSS.

```
#spoe {
--color: red;
}
[...]
<datapoint id="spoe">
```

Listing 5.2: Change the color with CSS.

```
.linechart {
--show-ticks: false;
}
[...]
<ec-linechart class="linechart">
```

Listing 5.3: Specify if ticks are shown.

```
<!--Other HTML-->
<style>
  .mlinechart1 {
    --chart-width: 500;
    --chart-height: 300;
    --show-ticks: false;
  }
  #samsung {
    --color: blue;
  }
  #apple {
    --color: green;
  }
</style>
<!--Other HTML-->
<ec-linechart class="mlinechart1">
  <table>
    <thead>
      <tr>
        <th id="samsung">Samsung</th>
        <th id="apple">Apple</th>
      </tr>
    </thead>
    <!--Other HTML-->
  </table>
</ec-linechart>
```

Listing 5.4: CSS Examlpe showing all possible properties.

Chapter 6

Limitations and Future Work

With every project there come some great features, but also there has to be some limitations. This is also the case for EasyCharts, therefore this will be discussed in the following chapter, together with some possible future work improving the library.

6.1 Limitations

In Chapter 2, the 2 different modes were discussed in order to input data: Data series mode and table mode. As it is basically working, it has the disadvantage, that - especially in table mode - the table is no longer correctly displayed, when turning JavaScript off. The reason for this, is that in the `<thead>` element one has to specify the line names, or some dummy string value, if having single data series. It would be better to leave the `<thead>` element for being the input for the x and y axis names and maybe specify a new element like for example: `<line-name>` or something similar like that. Maybe the internal data structure should be changed, to make this more flexible. The last approach would be to split the charts into separate files, because so far there is only one file, where all the source code lies, but maybe as this project is scaling there should be one file per chart, and everything should ultimately get bundled again into one final file, using Rollup, which the includes then into his HTML files.

The next limitation is the parsing of CSS. Currently 4 attributes are possible with CSS and media queries also probably won't work. There is maybe some other way to achieve this, but in the scope of this project this was not possible to evaluate so far.

6.2 Future Work

Future work for this project could be:

- Create other chart types: Pie Chart, Scatter Plot, Parallel Coordinates, etc.
- Make possible with multiple data series. Currently only single data series are possible with bar chart.
- Add more CSS: Create media query possibility, add more properties for charts (stroke width, font size, font type, etc.).
- Save the generated chart as an SVG. Maybe making some flag in CSS (`--save-chart: true`) and the image gets saved as an SVG.
- Add WAI-ARIA accessibility features to the SVG chart, could be done in cooperation with the WAI-ARIA survey topic.

Chapter 7

For Developers

The latter chapters focused more on the usage of this library. This chapter is devoted for possible future developers of EasyCharts. On the one hand the general structure of the project is discussed, but on the other hand also the building process gets analyzed.

7.1 Installation

To install the EasyCharts library, follow the three steps:

1. Clone or download the repositor from Github. (`https://github.com/dorianpercic/EasyCharts`).
2. Install and use the minimum NodeJS version v.14.18.1 (npm v6.14.15).
3. Install the required dependencies by running the following command in the project root directory: `npm install`. This installs all the development dependencies as well as source code dependencies.

7.2 Project Structure

The project is basically subclassified into two parts:

1. Source code in `src/` directory.
2. Generated code in `dist/` directory.

In the `src/` directory there is the source file called `easycharts.mts`, which is a TypeScript implementation of the library. In general this library is implemented as an ECMAScript module, allowing opportunities for tree shaking and improving the general execution speed. Furthermore, the `src/` folder contains the `examples/` folder, which gets copied to the `dist/` directory, providing some examples for the end user.

The `dist/` directory gets generated through the build process and it contains the transpiled source code file called `easycharts.js` and the already mentioned `examples/` folder. The general project structure can be seen in Listing 7.1.

7.3 Build Process

The project uses Gulp as the build tool. To build the project, run this command in the project root directory: `npx gulp`. This command does basically 3 things:

1. Copies the `src/examples` folder into `dist/` to provide some examples to the user.

```
EasyCharts
|-- .gitignore
|-- LICENSE
|-- README.md
|-- dist
|   |-- easycharts.js
|   |-- examples
|       |-- barchart
|       |   |-- barchart-table.html
|       |   |-- easycharts.js
|       |   |-- style.css
|       |-- linechart
|       |   |-- easycharts.js
|       |   |-- linechart-dataseries.html
|       |   |-- linechart-table.html
|       |   |-- style.css
|       |-- multilinechart
|           |-- easycharts.js
|           |-- multilinechart-dataseries.html
|           |-- multilinechart-table.html
|           |-- style.css
|-- gulpfile.mjs
|-- package-lock.json
|-- package.json
|-- rollup.config.mjs
|-- src
|   |-- easycharts.mts
|   |-- examples
|       |-- barchart
|       |   |-- barchart-table.html
|       |   |-- easycharts.js
|       |   |-- style.css
|       |-- linechart
|       |   |-- easycharts.js
|       |   |-- linechart-dataseries.html
|       |   |-- linechart-table.html
|       |   |-- style.css
|       |-- multilinechart
|           |-- easycharts.js
|           |-- multilinechart-dataseries.html
|           |-- multilinechart-table.html
|           |-- style.css
|-- tsconfig.json
```

Listing 7.1: Project structure showing all files and folders of EasyCharts.

2. Transpiles the source file `easycharts.mts` and it generates the JavaScript file `easycharts.js` into the `dist/` directory. The transpiliation works using Rollup, whereas it bundles the easycharts.mts with components of the D3 library.

3. Executes the `npm run serve` command creating a local server serving local project files, enabling the user to avoid the Cross Origin Policy. So the user can for instance input an external CSS file.

A video, showing the build process in more detail can be seen here: `https://youtu.be/VZyqQjwlz2c`.

Bibliography

MDN contributors [2023a]. *Using custom elements*. 22 Jun 2023. `https://developer.mozilla.org/en-US/docs/Web/API/Web_components` (cited on page 1).

MDN contributors [2023b]. *Web Components*. 20 Jun 2023. `https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements` (cited on page 2).