HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Bachelorarbeit

im Studiengang

Online Medien Bachelor

Prototypische Entwicklung von interaktiven und performanten Web-Komponenten zur Datenvisualisierung mit CSS Houdini

Referent	:	Prof. Dr. Dirk Eisenbiegler
Koreferent	:	Timo Mayer
Vorgelegt am	:	30.12.2021
Vorgelegt von	:	Christoph Saile Matrikelnummer: 260599 Lange Furche 37, 72072 Tübingen christoph.saile@googlemail.com

Abstract

Gegenstand dieser Ausarbeitung ist die Evaluierung der Eignung von CSS Houdini zur Darstellung von interaktiven und performanten Diagrammen im Web, im Vergleich zu etablierten Technologien für diesen Einsatzzweck. Dazu werden eine JavaScript-Library und eine Webanwendung, in welcher die entworfenen Diagramme visualisiert werden, entwickelt. Einleitend wird der Stand der Technik anhand von anerkannten Diagramm-Libraries untersucht und deren Technologien zur Visualisierung von Daten gegenübergestellt. Daraufhin wird die Architektur, die Projektstruktur und der Entwurf der angefertigten Komponenten beschrieben. Abschließend wird die Technologie CSS Houdini anhand der entwickelten Library evaluiert. Die Thematik und das Ergebnis der Bachelorarbeit ist für Interessenten aus dem Fachgebiet Webentwicklung von Bedeutung.

The subject of this elaboration is the evaluation of CSS Houdini for displaying interactive and performant diagrams on the web, in comparison to established technologies. For this purpose, a JavaScript library and a web application, in which the designed diagrams are visualized, will be developed. In the first part, the state of the art is analyzed by examining already established diagram libraries and comparing their technologies for the visualization of data. Thereupon the architecture, the project structure and the conception of the created components are described. Finally, the technology CSS Houdini is evaluated on the basis of the developed library. The topic and the outcome of the bachelor thesis are relevant for people who are interested in the field of web development.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

SVG	Scalable Vector Graphics
NPM	Node Package Manager
W3C	World Wide Web Consortium
CSSOM	CSS Object Model
CD	Continuous Deployment
CI	Continuous Integration
SEO	Search Engine Optimization
PWA	Progressive Web App
API	Application Programming Interface
HTML	Hypertext Markup Language
XML	Extensible Markup Language
CSS	Cascading Style Sheets
DOM	Document Object Model
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
WWW	World Wide Web

1 Einleitung

Daten können auf unterschiedliche Weise im Web visualisiert werden. Eine Form der Visualisierung von Daten sind Diagramme. Diagramme können im Web entweder als statische Grafiken oder als dynamische und interaktive Web-Komponenten dargestellt werden. Diese Facharbeit thematisiert die Erstellung von dynamischen und interaktiven Diagrammen im Web.

> *For many, many, many years [Adobe] Flash was the only and preferred solution for bringing animated, interactive elements to the Web*[1]

Das vorliegende Zitat zeigt, dass Adobe Flash jahrelang den Branchenstandard zur Generierung von interaktiven Elementen und somit auch von interaktiven Diagrammen im Web bildete. Aufgrund der zunehmenden Sicherheitslücken von Adobe Flash und der Einführung von Hypertext Markup Language (HTML)-5 nahm in den letzten Jahren der weltweite Anteil an Webseiten, welche Adobe Flash nutzen signifikant ab. Im Jahr 2021 nutzten lediglich 2,2% aller Webseiten Adobe Flash.[2] Mit dem Ende der Ära Adobe Flash kristallisierten sich neue Technologien zur Entwicklung von interaktiven Diagrammen als neue Branchenstandards heraus. Diese Entwicklung verdeutlicht, dass in der Webentwicklung ein fortlaufender Wandel an Technologien vorherrscht.
Eine dieser neuen Technologien ist Cascading Style Sheets (CSS) Houdini. Im Rahmen dieser Ausarbeitung wird überprüft, inwiefern sich CSS Houdini für die Erstellung von interaktiven und performanten Diagrammen zur Datenvisualisierung im Web eignet. Dazu wird eine JavaScript-Library und eine Webanwendung, in welcher die entworfenen Diagramme visualisiert werden, entwickelt. Mithilfe der entwickelten Library sollen mit einem geringem Aufwand neue Diagramme erstellt werden können. Bei der Entwicklung der Web-Komponenten liegt der Fokus auf der Funktionalität der Diagramme. Die visuelle Darbietung der Diagramme ist dabei zweitrangig.
Einleitend wird der Stand der Technik anhand von sechs anerkannten Diagramm-Libraries untersucht. Basierend auf den Ergebnissen der Analyse werden die verwendeten Technologien zur Visualisierung von Daten erläutert und deren Eigenschaften gegenübergestellt. Anschließend werden anhand der Gegenüberstellung der Technologien Kriterien für die Aufgabenstellung der Thesisarbeit abgeleitet. Daraufhin wird

[1] [Micb]
[2] [Bra]

die Umsetzung des Projekts beschrieben. Dabei wird zuerst der Begriff CSS Houdini definiert, um eine theoretische Grundlage zu schaffen. Danach werden die bei der Entwicklung der Web-Komponenten verwendeten Schnittstellen von CSS Houdini vorgestellt. In dem darauffolgenden Unterkapitel wird die Auswahl der Architektur der entwickelten JavaScript-Library „HoudiniCharts“ und der dazugehörigen Webanwendung vorgestellt. Dabei werden die Gründe und die Relevanz für die Verwendung der Programmiersprache TypeScript thematisiert. Weiterführende Informationen zu der Programmiersprache TypeScript sind in der Fachliteratur „Pro TypeScript - Application-Scale JavaScript Development“ von Steve Fenton zu finden.[3] Anschließend wird die Projektstruktur und der Entwurf der Web-Komponenten anhand von Klassendiagrammen beschrieben. Basierend auf dem generischen Aufbau der Diagramm-Klassen werden außerdem die Besonderheiten und Herausforderungen bei der Erstellung eines Linien- und Radardiagramms aufgezeigt. In Kapitel 5 wird die aus der Entwicklung hervorgegangene Library anhand der Kategorien Performance, Interaktivität und Barrierefreiheit evaluiert. Als Referenz für die Evaluation werden zwei bereits etablierte Diagramm-Libraries herangezogen. Zusätzlich wird in diesem Kapitel das entwickelte Testverfahren für die Messung der Performance beschrieben und die daraus resultierenden Messergebnisse ausgewertet. Abschließend erfolgt eine Zusammenfassung des im Rahmen der Ausarbeitung entstandenen Projekts und ein Ausblick auf die potenziell zukünftige Weiterentwicklung der entworfenen Library „HoudiniCharts“.

[3] [Fen17]

2 Stand der Technik

Um den aktuellen Stand der Technik festzuhalten, werden im folgenden Kapitel eine Auswahl der am häufigsten von der für Webentwickler bekannten Plattform `https://www.npmjs.com/` heruntergeladenen Diagramm-Libraries auf ihre technische Umsetzung untersucht. Dabei wird tabellarisch festgehalten auf welchen Technologien die jeweiligen Libraries aufbauen. Basierend auf dem Ergebnis der Untersuchung werden anschließend die Technologien gruppiert und näher erläutert. Am Ende des Kapitels werden die Eigenschaften der Technologien in Bezug auf die Erstellung von Diagrammen gegenübergestellt und miteinander vergleichen.

2.1 Analyse vorhandener Diagramm-Libraries

Das folgende Diagramm zeigt sechs der am häufigsten von der Plattform „npm“ heruntergeladenen Diagramm-Libraries innerhalb der letzten 5 Jahre.[4] Die Plattform „npm“ ist ein öffentliches Repository an Open Source Packages für den gleichnamigen Node Package Manager (NPM). Der NPM wird derzeit von rund 11 Millionen Entwicklern weltweit genutzt und besitzt mit mehr als einer Millionen Packages die größte Software-Registry der Welt. Aufgrund dessen ist der NPM der De-facto-Standard Package Manager für Frontend Entwicklung. Aus diesem Grund können seine Download-Statistiken als repräsentativ angesehen werden.[5]

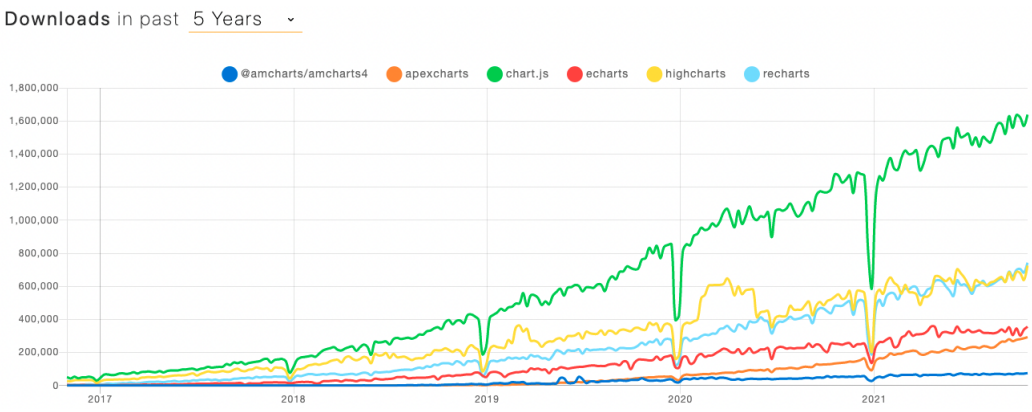


Abbildung 1: Downloads Diagramm-Libraries [Joh]

[4] [Joh]
[5] [npm]

Die Analyse der in Abbildung 1 aufgeführten Libraries ergibt, dass die Libraries „Highcharts“, „Recharts“, „ApexCharts“ und „amCharts“ Scalable Vector Graphics (SVG) als Technologie verwenden, um Diagramme darzustellen.[6] Die Library mit den meisten Downloads „Charts.js“ setzt hingegen auf das HTML „<canvas>“ Element zur Darstellung von Diagrammen.[7] Der Anbieter „ECharts“ stellt als einziger dem Anwender die Möglichkeit bereit, sowohl SVG- als auch Canvas-basierte Diagramme zu erstellen.[8]

Libraries	Technologie	
	SVG	Canvas
amCharts		
ApexCharts		
Chart.js		
ECharts		
Highcharts		
Recharts		

Tabelle 1: Gegenüberstellung Diagramm-Libraries

Das Ergebnis der Analyse der am meist genutzten Anbieter zeigt, dass ausschließlich die beiden Technologie SVG und Canvas den aktuellen Stand der Technik abbilden, um Diagramme im Web darzustellen.

2.2 SVG

Im Jahr 2001 wurden erstmals SVGs vom World Wide Web Consortium (W3C) als Sprache zum Beschreiben von 2D-Vektorgrafiken empfohlen. SVGs basieren auf Extensible Markup Language (XML) und werden heutzutage von allen gängigen Webbrowsern unterstützt. Aufgrund des XML-basierten Formates, lassen sich SVGs mithilfe eines Texteditors problemlos erstellen und bearbeiten. Ein Vorteil von SVGs im Vergleich zu anderen Formaten zur Darstellung von Grafiken, wie beispielsweise „JPG“ oder „PNG“, besteht darin, dass SVGs ohne Qualitätsverlust skalierbar sind. Darüber hinaus können SVGs neben dem Standard HTML „<img>“ Element und der CSS-Property „background-image“ auch inline eingebunden werden. Inline eingebundene SVG-Grafiken sind Teil des Document Object Model (DOM) und können mithilfe von

[6] [Hig, Rec, Ape, amC]
[7] [Cha]
[8] [Fou]

JavaScript oder CSS manipuliert und animiert werden. Zusätzlich sind sie für Suchmaschinen indexierbar und zugänglich für Screenreader.[9]
Neben dem Beschreiben von simplen geometrischen Formen eignen sich SVGs auch zum Erstellen von komplexen und interaktiven Diagrammen. Gründe hierfür sind die bereits genannte Skalierbarkeit und die Möglichkeit mit der Maus oder der Tastatur mit dem Diagramm zu interagieren.[10]
Libraries zum Erstellen von Diagrammen die auf SVG basieren, wie beispielsweise „amCharts“ oder „ApexCharts“, verwenden häufig intern oder extern entwickelte SVG-Engines um programmatisch SVGs zu erstellen.[11]

2.3 Canvas

Das HTML „<canvas>“ Element basiert auf einem Raster, welches aus Pixeln besteht. Die einzelnen Pixel des Rasters können mithilfe eines JavaScript Application Programming Interface (API) dynamisch manipuliert werden. Dadurch können vielschichtige Grafiken gezeichnet, Bilder bearbeitet oder Diagramme und Animationen erstellt werden. 2014 wurde das HTML „<canvas>“ Element mit der Einführung von HTML5 vorgestellt. Es wird aktuell von allen gängigen Browsern nativ unterstützt. Objekte, die auf das Canvas Element gezeichnet werden, sind nicht Teil des DOM, sondern existieren lediglich innerhalb des Kontexts des jeweiligen Elementes. Dadurch können den gezeichneten Objekten weder Event Handler noch Styles per CSS zugeordnet werden. Außerdem haben Screenreader keine Information über die gezeichneten Elemente innerhalb der Canvas-Fläche. Damit Objekte innerhalb eines Canvas-Elements gestyled oder durch Benutzerinteraktionen angesprochen werden können, wird zusätzlicher JavaScript Programmiercode benötigt. Da nicht jedes gezeichnete Objekt als ein Element im DOM gespeichert wird, ergibt sich außerdem der Vorteil, dass die Größe des DOM gering gehalten wird und Anweisungen performant durchgeführt werden können.[12]

[9] [Lib18, 3-7]
[10] [Lib18, 207-208]
[11] [amC, Ape]
[12] [LAS10, 25-28]

2.4 Gegenüberstellung SVG und Canvas

In diesem Abschnitt werden Eigenschaften von SVG-basierten Diagrammen im Vergleich zu Canvas-basierten Diagrammen aufgelistet und miteinander verglichen.
Aufgrund der Tatsache, dass SVG-Diagramme aus Vektoren bestehen, sind sie nativ skalierbar und responsiv.[13] Ein Canvas-Diagramm hingegen ist eine Bitmap-Grafik, welche aus Pixeln besteht. Bei einer Änderung des Viewports muss deshalb das Diagramm neu gezeichnet werden, um eine Responsivität zu gewährleisten.[14]
In Bezug auf die Barrierefreiheit sind Canvas-Diagramme, im Vergleich zu SVG-Diagrammen, ebenfalls im Nachteil. Der ausschlaggebende Grund hierfür ist, dass wie bereits in Kapitel 2.3 erläutert, die auf das Canvas gezeichneten Objekte nicht Teil des DOM sind. Infolgedessen sind diese Objekte nicht zugänglich für Screenreader. Es kann lediglich das Canvas-Element selbst mit einem ARIA Attribut versehen werden, um die Zugänglichkeit zu ermöglichen. Workarounds, wie das Nachbauen der Struktur im shadow DOM des Canvas-Elements mithilfe von HTML-Elementen verbessern die Barrierefreiheit, sind allerdings mit einem Mehraufwand verbunden.[15]
Die Elemente eines SVG-Diagramms wiederum sind Teil des DOM und somit nativ zugänglich für Screenreader. Dadurch können SVG-Diagramme mit wenig Aufwand barrierefrei gestaltet werden.[16]
Ein weiterer Vorteil, dass SVGs Teil des DOM sind besteht darin, dass die einzelnen Formen mithilfe von CSS-Animationen animiert und mit Event Handlern versehen werden können.[17] Bei Canvas-Diagrammen ist dies nicht möglich. Animationen und Interaktionen müssen mit zusätzlichem JavaScript Programmiercode realisiert werden.[18] Interaktionen können beispielsweise ermöglicht werden, indem die Koordinaten der aktuellen Mausposition mit den Koordinaten des jeweiligen Elementes abgeglichen werden.
Dadurch dass alle Element eines SVG-Diagramms im DOM gespeichert werden, führt dies bei Diagrammen mit großen Datenmengen zu einem DOM mit vielen Einträgen. Infolgedessen wird die benötigte Zeit um ein Dokument zu verarbeiten, höher. Bei Canvas-Diagrammen mit vielen Datensätzen bleibt die Größe des DOM gering.
Darüber hinaus sind Canvas-Diagramme Off-Thread fähig. Auf diese Weise können die Diagramme in einem separaten Thread erstellt werden und blockieren nicht den

[13] [Lib18, 3-7]
[14] [LAS10, 25-28]
[15] [Mica][LAS10, 25-28]
[16] [Lib18, 3-7]
[17] [Lib18, 3-7]
[18] [LAS10, 25-28]

Main-Thread.[19] SVG-Diagramme unterstützen diese Funktion nicht.
Zusätzlich ermöglicht das Canvas-API Pixel Manipulationen in Diagrammen. Dadurch können Bilder, welche in ein Canvas-Element geladen werden, bearbeitet werden.[20]

Eigenschaften	SVG	Canvas
Skalierbarkeit		
Responsivität		
Barrierefreiheit		
DOM Größe		
Off-Thread-Support		
Event Handler		
CSS-Animationen		
Pixel-Manipulation		

Tabelle 2: Gegenüberstellung SVG und Canvas

Anhand des Ergebnisses, der in Tabelle 2 visualisierten Gegenüberstellung, werden die folgenden Schlussfolgerungen getroffen. SVG-Diagramme können mit geringem Aufwand interaktiv und barrierefrei gestalten werden, während Canvas-Diagramme aufgrund ihrer Performance besonders für Diagramme mit einer hohen Anzahl an Datensätzen geeignet sind.

[19] [Gas]
[20] [LAS10, 57]

3 Aufgabenstellung

Das Ziel dieser Ausarbeitung ist die Evaluation der Eignung von CSS Houdini, zur Erstellung von performanten und interaktiven Diagrammen im Web. Dafür soll eine eigene JavaScript-Library mit dem Namen „HoudiniCharts“ entwickelt werden. Diese soll eine nutzbringende Alternative zu Canvas- und SVG-basierten Diagrammen darstellen. Der Vorteil der Library soll darin bestehen, dass die interaktiven Elemente eines Diagramms Teil des DOM sind und aufwändige Zeichenoperationen mithilfe von Paint-Worklets performant durchgeführt werden. Zusätzlich wird eine Anwendung in Form einer Website entwickelt, in der die Diagramme der Library in unterschiedlichen Kontexten visualisiert werden. Abschließend wird „HoudiniCharts“ mit bereits etablierten Diagramm-Libraries verglichen. Dabei stehen die Bereiche Performance, Interaktivität und Barrierefreiheit im Fokus. Bei der Entwicklung der Library wird auf die Modularität und Wiederverwendbarkeit der Komponenten geachtet, damit diese in Zukunft weiter ausgebaut werden können. Im Folgenden wird das Ziel und die Aufgabenstellung der Ausarbeitung genauer definiert und in die Bereiche Konfigurierbarkeit, Performance, Interaktivität und Barrierefreiheit unterteilt. Diese Bereiche werden gewählt, da sie relevante Faktoren einer Diagramm-Library darstellen. Eine ausreichende Konfigurierbarkeit ist essentiell für eine Diagramm-Library, damit der Nutzungskontext eines Diagramms dynamisch angepasst werden kann. Die Eigenschaften Performance und Interaktivität sind fundamental, um eine einwandfreie User Experience gewährleisten zu können. Der Faktor Barrierefreiheit ist ebenfalls wichtig, damit die Inhalte eines Diagramms uneingeschränkt von allen Benutzergruppen konsumiert werden können.

3.1 Konfigurierbarkeit

Der Benutzer soll per Konfigurations-Objekt die Möglichkeit haben, ein für sein Nutzungskontext angepasstes Diagramm erstellen zu können. Dies beinhaltet die Auswahl der Diagrammart, das Festlegen des Diagrammtitels und das Festlegen der Achsenbeschriftung. Außerdem soll der Benutzer die Einstellungsmöglichkeit haben, mittels des Konfigurations-Objekts, die darzustellenden Datensätze festzulegen. Die Diagramme müssen Daten im Text-, Zahlen- als auch Datumsformat darstellen können. Darüber hinaus soll die Option bestehen, eine benutzerdefinierte Skalierung zu erstellen, indem ein minimaler und ein maximaler Wert für die Skalierung festgelegt wird. Des

weiteren kann die Farbe der Gitternetzlinien und die Farben der einzelnen Datensätze definiert werden. Zuletzt kann der Benutzer festlegen, ob der Bereich unter einem Graphen ausgefüllt ist oder nicht. Für die abschließende Evaluierung der entwickelten Library sollen mindestens zwei Diagrammarten implementiert werden. Ein Linien- und ein Radardiagramm. Die Implementierung von Animationen und zusätzlichen Layout Optionen sind nicht Teil der Aufgabenstellung.

3.2 Performance

Die entwickelte Library soll hinsichtlich der Performance konkurrenzfähig gegenüber bereits etablierten Diagramm-Libraries sein. Dazu werden mithilfe der in Kapitel 4.1 beschriebenen APIs, Off-Thread fähige Worklets erstellt. Diese Worklets sollen dabei helfen den Inhalt der Diagramme performant darzustellen. Zusätzlich soll im Vergleich zu SVG-basierten Diagrammen, durch die Verwendung von Paint-Worklets, die Menge an Elementen innerhalb des DOMs reduziert werden.

3.3 Interaktivität

Interaktivität ist ein wichtiger Bestandteil des modernen World Wide Web (WWW). Aus diesem Grund sollen die Diagramme der entwickelten Library nicht nur Daten visualisieren, sondern diese interaktiv darstellen. Der Benutzer soll die Möglichkeit haben mit den einzelnen Datenpunkten der Diagramme zu interagieren. Wird über einen der Datenpunkte mit dem Mauszeiger gehovert, öffnet sich ein Tooltip mit konkreten Informationen zu dem jeweiligen Datenpunkt. Darüber hinaus können alle Datenpunkte per Klick mit einer Hilfslinie markiert werden. Über das Klicken auf einen der Datensätze innerhalb der Legende, kann der jeweilige Datensatz ein- und ausgeblendet werden. Die Diagramme sollen außerdem responsiv gestaltet sein, damit bei einer Anpassung des Viewports weiterhin alle Elemente lesbar bleiben.

3.4 Barrierefreiheit

Neben der Performance und Interaktivität steht auch die barrierefreie Gestaltung der Diagramme im Vordergrund. Dadurch soll die eigenständige Nutzung der Diagramme von Menschen mit einer Sehbehinderung oder mit eingeschränkten motorischen Fähigkeiten gewährleistet werden. Um dies zu berücksichtigen werden die Diagramme textlich beschrieben und für die Tastatursteuerung optimiert. Weitere Maßnahmen zur Optimierung der Barrierefreiheit, wie das Anpassen von Schriften, Kontrasten und Bedienoberflächen, sind nicht Teil der Aufgabenstellung. Im Vergleich zu SVG- oder

Canvas-basierten Diagrammen sollen die entwickelten Diagramme durch die Verwendung von semantisch korrekten HTML-Elementen, mit einem geringeren Aufwand, barrierefrei gestaltet werden können.

4 Umsetzung

Wie bereits in Kapitel 3 beschrieben ist das Ziel dieser Ausarbeitung die Eignung von CSS Houdini zur Darstellung von Diagrammen im Web zu evaluieren. Dazu soll eine eigene JavaScript Library entwickelt werden. Um den Begriff CSS Houdini theoretisch einzuordnen, wird dieser in diesem Kapitel zuerst definiert und anschließend der aktuelle Status der Technologie dargestellt. Dabei wird auf die Browserunterstützung von CSS Houdini eingegangen und die verwendeten APIs in einzelnen Unterkapiteln beschrieben. Daraufhin wird in Kapitel 4.2 die für die Umsetzung gewählte Architektur der Library und der dazugehörigen Web-Anwendung beschrieben. In diesem Rahmen werden auch die relevanten Kriterien für die Entscheidung der Architektur erörtert. Anschließend wird der Aufbau des Projekts anhand einer Abbildung der Verzeichnisstruktur erläutert. Zum Schluss werden die Komponenten und Worklets, welche für die Library „HoudiniCharts“ entwickelt wurden, mithilfe von Klassendiagrammen beschrieben.

4.1 CSS Houdini

Der Begriff CSS Houdini umfasst eine Sammlung von Browser APIs, die den Zugriff auf Teile der CSS-Engine des Browsers ermöglichen. Dadurch sollen Webentwickler mehr Kontrolle über das CSS Object Model (CSSOM) und somit über das Rendering einer Website bekommen. Infolgedessen können eigenständig neue CSS-Features entwickelt werden, ohne auf die native Implementierung in den Browsern warten zu müssen. CSS Houdini wird von der Houdini Taskforce entwickelt. Die Houdini Taskforce besteht aus mehreren Entwicklern von nennenswerten Unternehmen, wie beispielsweise Mozilla, Apple, Microsoft und Google. Aktuell befindet sich CSS Houdini noch in der Entwicklungsphase.[21]

[21] [Bec, Sura]

	Google Chrome	Microsoft Edge	Opera	Samsung Internet	Mozilla Firefox	Apple Safari	Spec
Engine	Blink				Gecko	WebKit	-
Paint API (Explainer Demos Article)	Shipped (Chrome 65) Details	Shipped (Edge 79) Details	Shipped (Opera 52) Details	Shipped (Internet 9.2) Details	Under consideration Details	In Development Details	Candidate Recommendation Spec
Properties & Values API (Demos Article)	Shipped (Chrome 78) Details	Shipped (Edge 79) Details	Shipped (Opera 65) Details	Shipped (Internet 12.0) Details	Under consideration Details	Partial support (Safari TP 67) Details	Working Draft Spec
Typed OM (Explainer Article)	Shipped (Chrome 66) Details	Shipped (Edge 79) Details	Shipped (Opera 53) Details	Shipped (Internet 9.2) Details	Under consideration Details	In Development Details	Working Draft Spec
Layout API (Explainer Demos)	Partial support (Canary) Details	Partial support (Canary) Details	Partial support (Developer) Details	No signal	Under consideration Details	Under consideration Details	First Public Working Draft Spec
AnimationWorklet (Explainer Demos Article)	Partial support (Chrome 71) Details	Partial support (Edge 79)	Partial support (Opera 58)	Partial support (Internet 10.2)	No signal	Under consideration Details	First Public Working Draft Spec
Parser API (Explainer)	No signal	No signal	No signal	No signal	No signal	No signal	Proposal Spec
Font Metrics API (Explainer)	No signal	No signal	No signal	No signal	No signal	No signal	Proposal Spec

Abbildung 2: CSS Houdini Browser Support [Surb]

Der gegenwärtige Status der Entwicklung und der Browserunterstützung von CSS Houdini wird in Abbildung 2 veranschaulicht. Zum Zeitpunkt der Erstellung dieser Ausarbeitung werden drei von sieben APIs nativ von Chromium-basierten Browsern unterstützt. Chromium ist eine weit verbreitete Open Source Browser Engine, welche zum Beispiel von den Browsern Google Chrome, Microsoft Edge und Opera verwendet wird. Aufgrund des aktuellen Status der Entwicklung werden im Folgenden ausschließlich das „Paint API“, das „Properties and Values API“ und das „Typed Object Model API“ näher beschrieben.[22]

4.1.1 Paint API

Das Paint API wird nativ von allen Chromium-basierten Browsern unterstützt und kann unter Verwendung eines Polyfills auch in den Browsern Safari und Mozilla Firefox verwendet werden. Es basiert auf einem Teil des „2DRenderingContexts“ und ermöglicht es mittels JavaScript Programmiercode, Bitmap-Grafiken sowohl in den Hintergrund als auch in den Rahmen eines HTML-Elementes zu zeichnen. Zu beachten ist, dass sich die Koordinate (0/0) im Canvas Kontext oben links befindet. Somit muss das Vorzeichen des Wertes der y-Koordinate negativ sein, um eine Linie von Punkt (0/0) nach unten zu zeichnen.

[22] [Surb]

Der „2DRenderingContext“ ist Teil des HTML5 Canvas API. Um die Paint API nutzen zu können, muss zuerst ein Paint-Worklet erstellt werden.[23]
Worklets sind JavaScript-Module mit einem eigenen Scope. Sie haben daher wenig bis keinen Zugriff auf den globalen Kontext einer Anwendung. Sobald ein Worklet benötigt wird, wird es über die Rendering-Engine des Browsers aufgerufen. Mehrere Worklet-Prozesse können gleichzeitig und separat vom Main-Thread einer Anwendung ausgeführt werden. Dadurch können beispielsweise anspruchsvolle Zeichenanweisungen ausgeführt werden, ohne dabei den Main-Thread zu blockieren.[24]
Nachdem ein Paint-Worklet erstellt wurde, kann es registriert werden indem der gewünschte Name und die Klasse des Worklets an die „registerPaint“ Funktion übergeben werden. Abschließend kann das Worklet geladen und über die „paint“-Funktion aufgerufen werden. Die „paint“-Funktion kann in jeder CSS-Property verwendet werden, welche den Typ „<image>“ unterstützt. Mithilfe von CSS-Variablen können zusätzliche Argumente an das Paint-Worklet übergeben werden.[25]

4.1.2 Properties and Values API

Grundsätzlich werden alle CSS-Variablen vom Browser als String geparsed und als valide interpretiert. Dies lässt sich darauf zurückführen, dass der Browser nicht wissen kann, an welcher Stelle eine CSS-Variable verwendet werden soll. Dadurch besteht das Risiko, dass CSS-Variablen in einem falschen Kontext genutzt werden können. Ist eine CSS-Variable in einem falschen Kontext genutzt worden, wird das CSS-Statement nicht invalide, sondern bekommt den initialen oder vererbten Wert der Property zugewiesen.[26]
Um dieses Problem zu lösen, wurde das Properties and Values API entwickelt. Mithilfe des API können CSS-Variablen um die folgenden Eigenschaften erweitert werden:

- name: string
- syntax: string
- inherits: boolean
- initalValue: string

[23] [KJ]
[24] [Bec]
[25] [KJ]
[26] [Moz]

Das „Properties and Values API“ bietet dadurch die Möglichkeit einer direkten Integration mit dem Parser, wodurch Entwickler ihre eigenen CSS-Properties festlegen können. Die Properties können entweder direkt in einer CSS-Datei oder mit der JavaScript-Methode „registerProperty“ registriert werden.[27]

4.1.3 Typed Object Model API

Das Typed Object Model API verbessert den Zugriff mit JavaScript auf die CSS Eigenschaften eines HTML-Elementes. Mithilfe der API werden die Eigenschaften nicht mehr in Form eines Strings, sondern in Form eines „CSSUnitValue“-Objektes dargestellt. Das „CSSUnitValue“-Objekt besteht aus zwei Keys. Der erste Key ist der Wert und der zweite Key die dazugehörige Einheit der CSS-Property. Durch die neue Organisation und Strukturierung der Daten können diese schneller geparsed und einfacher mit JavaScript manipuliert werden. Außerdem entsteht durch die Typisierung ein Error-Handling, wodurch Fehler in der Entwicklung minimiert werden. Neben dem Abfragen von Styles bietet das API auch Methoden um die Styles eines HTML-Elementes zu verändern, entfernen, überprüfen oder neue Eigenschaften anzuhängen.[28]

4.2 Architektur

In diesem Unterkapitel wird die Architektur der entwickelten Library zur Darstellung von Diagrammen im Web veranschaulicht. Dabei wird einleitend die verwendete Programmiersprache beschrieben. Aufbauend auf einer Umfrage, wird anschließend die Relevanz der Programmiersprache offengelegt. Die Basis für das Entwickeln und das produktionsfertige Bündeln der Library bildet das in Kapitel 4.2.2 beschriebenen Tool Webpack. Abschließend wird neben der Architektur der Library auch die Distributions-Architektur der dazugehörigen Web-Anwendung, zum Visualisieren der entwickelten Diagramme, erläutert. Bei der Entwicklung der Library wird bewusst darauf geachtet, möglichst keine zusätzlichen JavaScript-Frameworks oder -Libraries zu verwenden, damit die Library weitestgehend unabhängig von Drittanbietern ist.

[27] [Bec]
[28] [ABR]

4.2.1 TypeScript

Die von Microsoft entwickelte Programmiersprache TypeScript wurde 2012 erstmals veröffentlicht, um das Entwickeln mit JavaScript effizienter zu gestalten.[29]

> *TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.*[30]

Bei TypeScript handelt es sich um eine objektorientierte Programmiersprache, welche mittels eines Compilers in JavaScript Programmiercode kompiliert wird. Der Programmiercode kann nach dem Kompilieren in jeder Umgebung ausgeführt werden, die JavaScript unterstützt. Dazu gehört beispielsweise Node.js oder ein Web-Browser. Einer der Vorteil von TypeScript gegenüber JavaScript besteht darin, dass bereits bei dem Kompilieren des Programmiercodes, Typ-Fehler festgestellt werden können.[31]

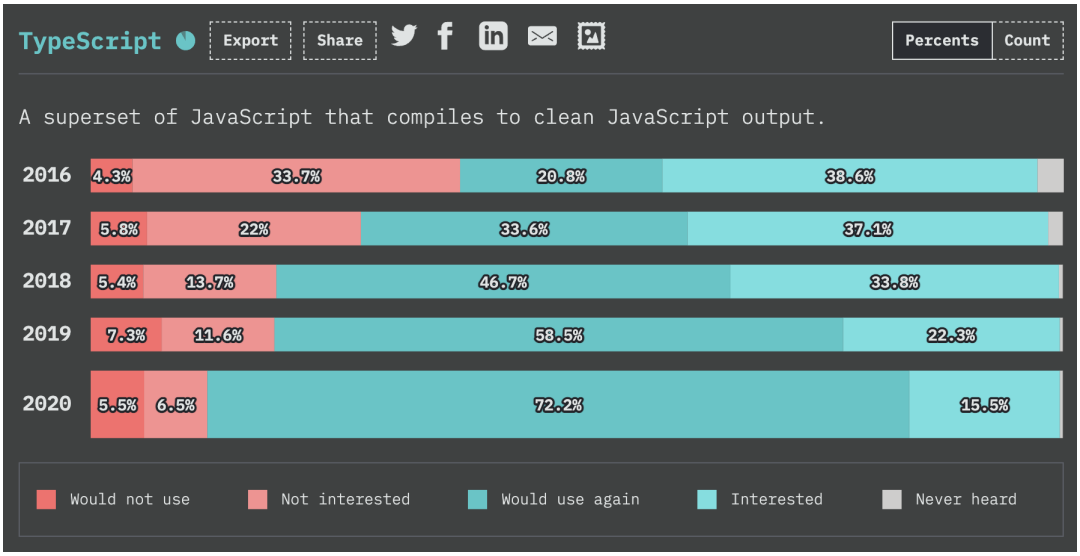


Abbildung 3: State of JavaScript 2020 - TypeScript [GB]

Wie in Abbildung 3 zu erkennen ist, haben immer mehr Web-Entwickler innerhalb der letzten fünf Jahre die Programmiersprache TypeScript verwendet. 2020 gaben 78% der Befragten an TypeScript zu benutzen und 93% bestätigten, dass sie erneut TypeScript benutzen würden. Die Abbildung stammt aus der jährlich durchgeführten Umfrage „State of JS“. Im letzten Jahr haben über 23.000 Entwickler an der Umfrage teilgenommen.[32]
Aufgrund der steigenden Akzeptanz von TypeScript kann davon ausgegangen werden, dass ausreichend Hilfestellungen in Foren und Plattformen, zum Austausch von

[29] [Micb]
[30] [Micb]
[31] [Fen17, 1-2]
[32] [GB]

Wissen, zur Verfügung stehen. Darüber hinaus ergibt sich aus dieser Tatsache die Möglichkeit, Lösungen für komplexe Problemstellungen kollaborativ zu erarbeiten. Zusätzlich ist TypeScript Lehrstoff in mehreren Veranstaltungen des Studiengangs Online Medien Bachelor an der Hochschule Furtwangen und ist somit von aktueller Relevanz.
Abschließend kann zusammengefasst werden, dass der Einsatz der Programmiersprache, aufbauend auf den genannten Argumenten, für die Entwicklung einer Library zur Darstellung von Diagrammen im WWW, geeignet ist. Neben TypeScript wird vereinzelt auch die Programmiersprache JavaScript verwendet. Grund hierfür sind die in Kapitel 4.1 vorgestellten APIs, welche sich aktuell noch in der Entwicklungsphase befinden und deshalb noch nicht vollständig von TypeScript unterstützt werden.

4.2.2 Webpack

Webpack ist ein unter Web-Entwicklern populärer Module Bundler, der es ermöglicht mehrere JavaScript-Module zu einer Datei zu bündeln. Dazu verwendet Webpack einen Dependency-Graphen, der alle in einem Input definierten Module analysiert und in ein Output-Paket zusammenfasst. Dadurch muss beim Aufruf einer Website nur eine JavaScript-Datei angefragt werden, anstellen von mehreren. Webpack hat weit aus mehr Vorteile als das Vereinen von JavaScript-Modulen. Aufgrund einer Vielzahl an Plugins kann der Funktionsumfang von Webpack deutlich erweitert werden. Auf diese Weise können beispielsweise CSS- und HTML-Dateien gebündelt oder TypeScript in JavaScript kompiliert werden. Darüber hinaus wird Webpack auch als Task-Runner verwendet, um Dateien zu komprimieren oder um einen lokalen Server für die Entwicklung zu hosten. Dadurch kann zum einen die Entwicklung spürbar vereinfacht und zum anderen die Größe der Dateien deutlich reduziert werden. Außerdem lassen sich für Webpack mehrere Konfigurationen festlegen und somit verschiedene Build- und Entwicklungs-Prozesse für ein Projekt generieren. Infolgedessen kann sowohl ein Build-Prozess für die Produktion der Library, als auch für die Produktion der Webanwendung erstellt werden. Für die Webanwendung müssen zusätzlich alle Dateien, die sich in dem Verzeichnis „public“ befinden, kopiert werden. Um Webpack verwenden zu können wird die Laufzeitumgebung Node.js benötigt.[33]
Laut der Umfrage „State of JS 2020“ ist Webpack mit 89% das am meisten verwendete Build Tool, gefolgt von Gulp mit 65%.[34] Aufgrund der bereits vorhandenen Erfahrungen mit Webpack und der großen Anzahl an Nutzern, ist Webpack somit optimal für die unter Kapitel 3 definierte Aufgabe, geeignet.

[33] [Igl19, 153-157]
[34] [GB]

4.2.3 Netlify

Das Cloud Computing Unternehmen Netlify wurde 2014 gegründet. Es bietet auf der gleichnamigen Plattform `https://www.netlify.com/` Dienstleistungen zum Generieren, Ausliefern und Hosten von Webanwendungen an. Mithilfe der Continuous Integration (CI)/Continuous Deployment (CD) Infrastruktur von Netlify lassen sich Projekte vollständig automatisiert veröffentlichen. Darüber hinaus können auch automatisierte Tests durchgeführt und Uniform Resource Locators (URLs) zur Preview generiert werden. Netlify bietet, neben kostenpflichtigen Abo-Modellen, auch ein kostenloses Starterpaket an.[35]
Netlify eignet sich bestens für die Distribution und das Hosting des in Kapitel 3 definierten Vorhabens, da die CI/CD-Infrastruktur modern und einfach zu bedienen ist und außerdem ein kostenloses Starterpaket zur Verfügung steht. Darüber hinaus konnten bereits im Vorfeld eigene Praxiserfahrungen mit dem Anbieter gesammelt werden. Die Webanwendung ist unter der Domain `https://houdini-charts.netlify.app/` aufzufinden.
Für die Einrichtung der CI/CD-Infrastruktur müssen bestimmte Schritte durchgeführt werden. Zuerst wird das gewünschte GitHub-Repository über ein Webinterface mit Netlify verbunden. Danach muss der Build-Befehl und das zu veröffentlichende Verzeichnis angegeben werden. Abschließend muss der Branch des Repositories und die Domain ausgewählt werden, welche für die CI/CD Infrastruktur verwendet werden sollen. Nachdem der Prozess zur Automatisierung erfolgreich aufgesetzt ist, wird sobald eine Änderung auf dem in der Konfiguration angegeben Branch erfolgt, der Build-Prozess angestoßen und die hinterlegte Website neu publiziert.

[35] [Net]

4.3 Projektstruktur

Die in Abbildung 4 dargestellte Grafik zeigt den strukturellen Aufbau der Library „HoudiniCharts“. Nachfolgend werden die Funktionalitäten und die Inhalte, der in der Grafik dargestellten Ordner, erläutert und beschrieben.

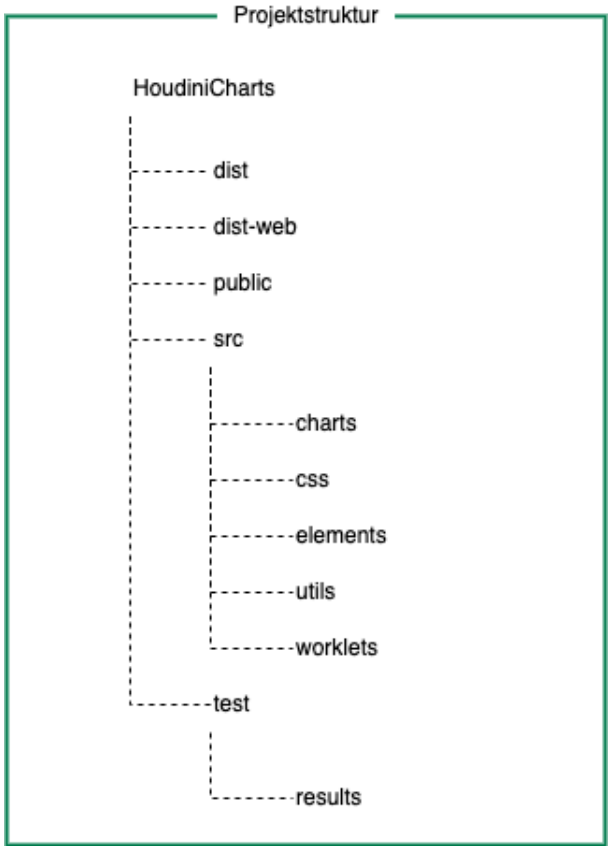


Abbildung 4: Projektstruktur - HoudiniCharts

Innerhalb des Stammordners „HoudiniCharts“ befinden sich alle relevanten Dokumente, welche für die Umsetzung der unter Kapitel 3 definierten Aufgabenstellung erstellt werden. Das Verzeichnis „dist“ enthält die produktionsfertigen Dateien der Library „HoudiniCharts“. Dazu zählen die mit dem Tool Webpack kompilierten, gebündelten und komprimierten JavaScript- und CSS-Dateien der Library. Im Ordner „dist-web“ befinden sich ebenfalls produktionsfertige Dateien, allerdings nicht für die Library „HoudiniCharts“, sondern für die entwickelte Webanwendung zur Darstellung und zum Vergleich der erstellten Diagramme. Der Inhalt des Verzeichnisses „dist-web“ wird über den in Kapitel 4.2.3 beschriebenen Anbieter Netlify unter der Adresse `https://houdini-charts.netlify.app/` veröffentlicht. Innerhalb des Ordners „public“ befinden sich zusätzliche Dateien für die Webanwendung. Dieser umfasst die Scripte von weiteren Diagramm-Libraries, als auch Mock-, HTML- und CSS-

Dateien. Das Verzeichnis „src“ beinhaltet mehrere Unterverzeichnisse in denen sich der Quellcode für die Entwicklung der Library befindet. In dem Unterordner „charts“ sind die Komponenten der jeweiligen Diagrammtypen. Weitere Komponenten, die eine Diagramm-Klasse erweitern, befinden sich in dem Verzeichnis „elements“. Zusätzliche Helfer-Funktionen werden in dem Ordner „utils“ gespeichert. Die CSS-Dateien werden in dem Verzeichnis „css“ abgelegt. Worklets, welche für die Diagramm-Library entwickelt werden, befinden sich in dem gleichnamigen Ordner „worklets“. Im letzten Verzeichnis „test“ befinden sich mehrere Node.js-Scripte zum Durchführen und Auswerten von Performancemessungen. Die Ergebnisse der Messungen und Auswertungen werden ebenfalls in diesem Verzeichnis abgelegt.

4.4 Komponenten-Entwurf

Das vorliegende Kapitel befasst sich mit den programmierten Komponenten und Worklets, aus denen sich die entwickelte JavaScript-Library „HoudiniCharts“ zusammensetzt. Dabei wird nach dem Top-Down-Prinzip zuerst die oberste Klasse im Call-Tree erläutert und anschließend auf die Klassen und Objekte eingegangen, welche davon ausgehend aufgerufen werden. Dementsprechend wird in diesem Unterkapitel zuerst die Klasse „HoudiniCharts“ erläutert. Anschließend wird der Aufbau des Konfigurations-Objekts zum Anpassen der Diagramme anhand einer Grafik visualisiert. Danach wird der allgemeine Aufbau der Diagramm-Klassen skizziert. Basierend auf dem generischen Aufbau der Diagramm-Klassen werden die Besonderheiten und Herausforderungen bei der Erstellung eines Linien- und Radardiagramms aufgezeigt und in separaten Abschnitten beschrieben. Zum Schluss werden die Komponenten „Header“, „Tooltip“ und „Accessibility“, die fester Bestandteil jeder Diagramm-Komponente sind, anhand von Klassendiagrammen dargestellt und erläutert.

4.4.1 HoudiniCharts

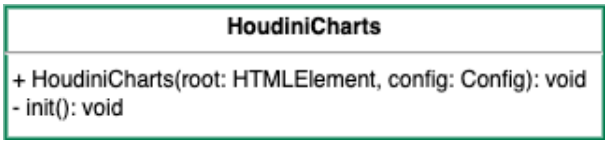


Abbildung 5: Klassendiagramm - HoudiniCharts

Die oberste Klasse im Call-Tree ist die in Abbildung 5 dargestellte Komponente „HoudiniCharts“. Die Klasse „HoudiniCharts“ stellt die Basis der gleichnamigen Library dar und benötigt zwei Attribute damit sie initialisiert werden kann. Der erste Parameter der Klasse ist vom Typ „HTMLElement“ und beschreibt den Container, in welchem

die Komponente gerendert werden soll. Der zweite Parameter ist ein Objekt vom Typ „Config“, welches in Abbildung 6 näher beschrieben wird. Das „Config“-Objekt definiert die Eigenschaften und das Aussehen eines Diagramms. Im Konstruktor der Klasse „HoudiniCharts“ wird zuerst die Methode „init“ aufgerufen. Innerhalb der „init“-Methode überprüft ein Switch-Statement, anhand des als Parameter übergebenen Konfigurations-Objekts, um welchen Diagrammtypen es sich handelt. Abhängig von dem jeweiligen Diagrammtypen werden die Parameter der Klasse an die entsprechende Diagramm-Komponente übergeben und eine Instanz von ihr gebildet. Stimmt der von dem Benutzer eingegebene Diagrammtyp nicht mit den vorhandenen Diagrammtypen überein, wird eine Fehlermeldung in der Konsole ausgegeben.

4.4.2 Konfiguration

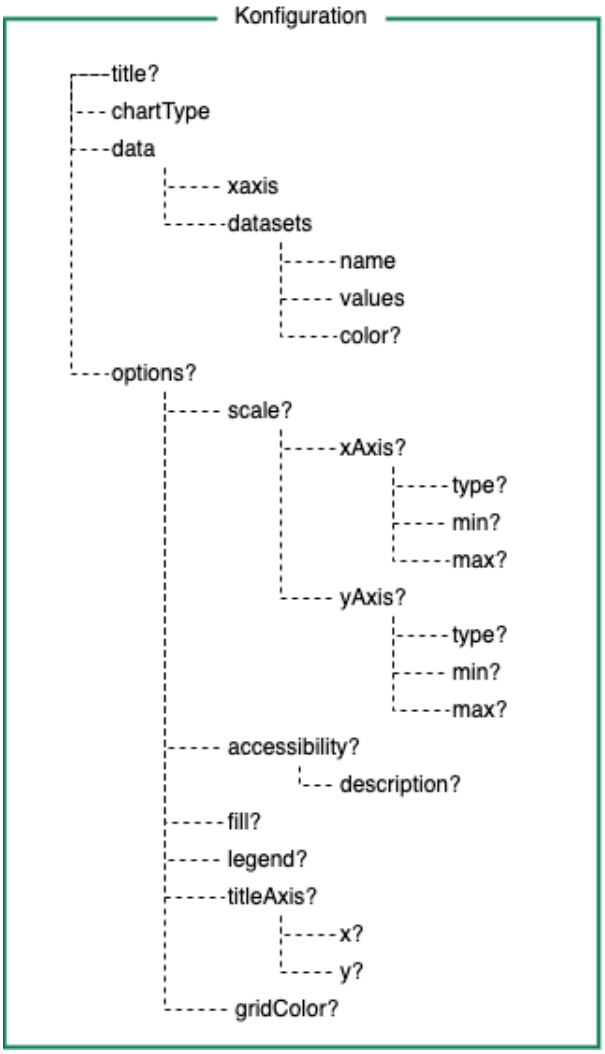


Abbildung 6: Aufbau „Config“-Objekt

Der Aufbau eines Konfigurations-Objektes ist in Abbildung 6 visualisiert. Das Konfigurations-Objekt definiert die wesentlichen Eigenschaften eines Diagramms. Jedes Diagramm benötigt ein Konfigurations-Objekt, damit es initialisiert werden kann. Im Folgenden wird der Aufbau des Objekts beschrieben und auf die einzelnen Eigenschaften eingegangen.
Die Schlüssel des Objektes werden durch die in Abbildung 6 zu sehenden Textfelder repräsentiert. Befindet sich ein Fragezeichen hinter einem der Objektschlüssel ist die jeweilige Einstellung optional und wird nicht für die Initialisierung der Klasse „HoudiniCharts“ benötigt. Die optionale Einstellung „title“ ermöglicht es dem Nutzer einen Titel für das Diagramm festzulegen. Dieser wird standardmäßig zentriert über dem Diagramm ausgerichtet. Der Objektschlüssel „chartType“ definiert die Art des Diagramms. Aktuell kann zwischen einem Linien- und einem Radardiagramm ausgewählt werden. Über das Feld „data“ kann die Beschriftung der x-Achse und die Anzahl an Datenserien festgelegt werden. Jede Datenserie besteht aus einem Namen, einer optionalen Farbe und einem Array mit den Werten der einzelnen Elemente. Die Koordinaten eines Datenpunktes werden über die Position innerhalb des Arrays und dem dazugehörigen Wert bestimmt. Weitere Eigenschaften des Diagramms können über den Objektschlüssel „options“ definiert werden. Dazu zählt unter anderem das Aktivieren der Diagramm Legende und das Aktivieren des barrierefreien Modus. Der barrierefreie Modus ermöglicht es, dass ein Diagramm zugänglich für Screenreader ist und der Inhalt eines Diagramms problemlos per Tastaturbefehle erreicht werden kann. Darüber hinaus kann über das Feld „scale“ der Typ und die Skalierung der Achsen eines Diagramms angepasst werden. Ist weder ein Minimum noch ein Maximum für die Skalierung definiert, wird die Skalierung automatisch anhand der Datensätze bestimmt. Zusätzlich kann ausgewählt werden, ob die gezeichnete Fläche eines Diagramms ausgefüllt werden soll oder nicht. Außerdem hat der Nutzer die Möglichkeit die Farbe der Gitternetzlinien zu bestimmen. Möchte der Benutzer einen Titel für die x- und y-Achse definieren, kann er dies über die Property „titleAxis“ festlegen. Alle Einstellungsmöglichkeiten, die in dem Objekt „options“ aufgeführten sind, können optional ausgewählt werden.

4.4.3 Allgemeiner Aufbau der Diagramm-Komponenten

Dieses Unterkapitel beschreibt den grundlegenden Aufbau einer Diagramm-Komponente und dient als Vorlage zur Implementierung von weiteren Diagrammarten. Aufgrund der Tatsache, dass die meisten Diagrammtypen mindestens eine Achse mit einer Skalierung besitzen, orientiert sich die dargestellte Struktur an dieser Richtlinie. Diagramme, die mehrere oder keine Achsen besitzen, müssen in ihrem Aufbau angepasst

werden. Die grundlegenden Methoden einer Diagramm-Komponente sind:

- „initScale“
- „initRender“
- „initStyles“
- „initEvents“

Wie bereits in Kapitel 4.4.1 beschrieben, benötigt jede Diagramm-Komponente die Attribute „root“ und „config“, damit sie initialisiert werden kann. Nach der Initialisierung wird zuerst im Konstruktor der Klasse die Methode „loadWorklets“ aufgerufen. Mithilfe der „loadWorklets“-Methode werden alle benötigten Paint-Worklets der Diagramm-Klasse asynchron geladen. Anschließend wird die Methode „init“ aufgerufen. Innerhalb dieser Methode werden die zuvor aufgelisteten, grundlegenden Methoden einer Diagramm-Komponente aufgerufen und nacheinander ausgeführt. Aufgrund der Komplexität des unter Kapitel 4.4.2 beschrieben Konfigurations-Objekts, werden außerdem alle Schlüssel des Objektes in einer separaten Property gespeichert. Dadurch bleibt der Programmiercode einer Diagramm-Komponente übersichtlich und strukturiert. Außerdem können innerhalb einer Property direkt Abfragen durchgeführt werden, um festzustellen, ob die entsprechende Option vom Benutzer definiert wurde oder nicht. Je nach Ergebnis der Abfragen kann anschließend ein Basiswert definiert werden. Ist beispielsweise kein Farbwert für die Gitternetzlinien festgelegt, wird der Property „configGridColor“ die Farbe Schwarz als Basiswert zugewiesen.
Die zuvor genannten grundlegenden Methoden einer Diagramm-Komponente werden im Folgenden in einzelnen Abschnitten näher erläutert. Dabei wird der Programmiercode zur Erstellung einer Diagramm-Komponente dargestellt und die bei der Umsetzung entstandenen Herausforderungen thematisiert.

„initScale“

Im folgenden Abschnitt werden die Properties und Funktionen beschrieben, welche für die Erstellung einer dynamischen Achsenskalierung benötigt werden. Die Herausforderung bei der Erstellung einer Achsenskalierung besteht darin, ein passendes Intervall zu finden, das mit der niedrigsten Zahl der Skalierung beginnt und mit der höchsten Zahl der Skalierung endet. Die Grundlage für die Berechnung einer solchen Skalierung bilden die beiden Properties „min“ und „max“. Sie legen den kleinsten und größten Wert der Achse fest. Um diese Werte zu bestimmen wird zuerst überprüft, ob in der Konfiguration unter dem Feld „scales“ eine benutzerdefinierte Skalierung festgelegt

wurde. Ist dies der Fall, entspricht das Minimum und das Maximum der Achse diesen Werten. Anderenfalls werden die beiden Werte über die Helfer-Funktionen „getMinValue“ und „getMaxValue“, anhand des höchsten und niedrigsten Wertes innerhalb der Datensätze, bestimmt.
Nachdem die Properties „min“ und „max“ bestimmt wurden, können anschließend die Größen für die Property „niceNumbers“ ausgerechnet werden. Dazu wird der kleinste und der größte Wert der Achse an die Funktion „calculateNiceScale“ übergeben. Innerhalb der Funktion wird der sogenannte „Nice Numbers“-Algorithmus ausgeführt. Der Begriff „Nice Numbers“ beinhaltet die Dezimalzahlen 1, 2 und 5, sowie alle Zehnerpotenzen dieser Zahlen. Die soeben genannten Zahlen werden im Allgemeinen als schöne Zahlen wahrgenommen. Der Algorithmus berechnet dynamisch, anhand der übergebenen Parameter, die Basiswerte, um eine nachvollziehbare und anschauliche Skalierung zu erstellen. Zu den Basiswerten zählt die niedrigste und höchste „Nice Number“, sowie die Größe des Abstandes zwischen den Intervallen der Skalierung. Im Folgenden wird die Funktion „niceNum“ des „Nice Numbers“-Algorithmus vorgestellt.[36]

```
function niceNum(localRange: number, round: boolean) {
  var n;
  var fraction;
  var niceFraction;
  n = Math.floor(Math.log10(localRange));
  fraction = localRange / Math.pow(10, n);
  if (round) {
    if (fraction < 1.5) niceFraction = 1;
    else if (fraction < 3) niceFraction = 2;
    else if (fraction < 7) niceFraction = 5;
    else niceFraction = 10;
  } else {
    if (fraction <= 1) niceFraction = 1;
    else if (fraction <= 2) niceFraction = 2;
    else if (fraction <= 5) niceFraction = 5;
    else niceFraction = 10;
  }
  return niceFraction * Math.pow(10, n);
}
```

Listing 4.1: Ausschnitt „Nice Numbers“-Algorithmus

[36] [Gla90, 61-63]

Die Funktion „niceNum“ besitzt die beiden Parameter „localRange“ und „round“, sowie die drei Variablen „n“, „fraction“ und „niceFraction“. In der 5. Zeile des Algorithmus wird zuerst der Wert von „n“ berechnet. Dazu wird der Exponent zur Basis 10 des Parameters „localRange“ bestimmt und auf den nächst größten Integer abgerundet. Anschließend wird in der darauf folgenden Zeile der Wert der Variablen „fraction“ berechnet. Dabei wird der Parameter „localRange“ durch die Potenz 10^n dividiert. Anhand der Größe der Variablen „fraction“ wird mithilfe von mehreren If-Statements, der Wert von „niceFraction“ bestimmt. Die Variable „niceFraction“ entspricht entweder dem Integer 1, 2, 5 oder 10. Zum Schluss wird der Wert der Variablen „niceFraction“ mit der Potenz 10^n multipliziert und an die Funktion zurückgegeben.
Die Funktion „niceNum“ wird zuerst für die Berechnung der Differenz zwischen Minimum und Maximum aufgerufen. Der Rückgabewert der Funktion wird in der Variablen „range“ gespeichert. Nachdem die Differenz ermittelt wurde, wird die Funktion erneut aufgerufen, um die Größe des Intervallabstandes zu berechnen und diese in der Variablen „tickSpacing“ zu speichern. Als Attribut wird der Quotient aus dem Wert der zuvor bestimmten Variablen „range“ und der gewünschten Anzahl an maximalen Intervallen übergeben. Nachdem die Werte der Variablen „range“ und „tickSpacing“ bestimmt wurden, kann anschließend das „Nice Number“-Maximum und -Minimum der Achsenskalierung berechnet werden. Für die Berechnung werden die folgenden beiden Formeln verwendet.

$$niceMin = \lfloor min/intervallabstand \rfloor * intervallabstand \tag{4.1}$$

$$niceMax = \lceil max/intervallabstand \rceil * intervallabstand \tag{4.2}$$

Zum Schluss werden die Variablen „tickSpacing“, „niceMin“ und „niceMax“ einem Objekt vom Typ „NiceNumbers“ hinzugefügt und als Rückgabewert an die Funktion „calculateNiceScale“ zurückgeliefert.
Mithilfe des Objekts kann anschließend die Differenz zwischen dem neu errechneten Maximum und Minimum der Skalierung bestimmt und in der Property „range“ der jeweiligen Diagramm-Komponente gespeichert werden. Außerdem lässt sich die Anzahl an Intervallen berechnen. Dazu wird die Variable „range“ durch den Wert des Objektschlüssels „tickSpacing“ dividiert.

„initRender“

Dieser Abschnitt beschäftigt sich mit der Generierung der Grundstruktur eines Diagramms. Damit die Struktur dynamisch erstellt werden kann, werden die HTML-Elemente mithilfe von Template-Strings erstellt und über die Methode „insertAdjacentHTML“ an die entsprechende Stelle innerhalb des DOM gerendert. Die Erstellung der Struktur eines Diagramms wird mit dem Aufruf der Methode „initRender“ initialisiert. Innerhalb dieser Methode werden weitere Methoden aufgerufen, die für das Erstellen und das Rendern der einzelnen Komponenten eines Diagramms zuständig sind.
Zuerst wird die Methode „renderWrapper“ aufgerufen. Der Aufruf der Methode führt dazu, dass ein HTML-Element mit dem Namen des Diagrammtyps in das „root“-Element der Komponente eingefügt wird. Alle folgenden HTML-Elemente werden in diesem Wrapper gerendert. Der Kopfbereich des Diagramms wird über die Methode „renderHeader“ erstellt. Innerhalb der „renderHeader“-Methode wird die in Kapitel 4.4.6 beschriebene Klasse „Header“ aufgerufen. Danach wird der Bereich für die Zeichenfläche und der Bereich für die Achse des Diagramms erstellt. Basierend auf den Werten der zuvor berechneten Property „niceNumbers“ kann die Anzahl und der Inhalt der benötigten HTML-Elementen für die Skalierung der Achse ermittelt und gerendert werden. Mithilfe der Methode „renderDatasets“ wird für jede in der Konfiguration definierte Datenserie ein HTML-Element erstellt. Anschließend werden innerhalb der Methode „renderDatapoints“ alle Datenserien mit den dazugehörigen Datenpunkten, in Form eines HTML „<button>“ Elements, befüllt. Zum Schluss wird über die Methode „renderTooltip“ der in Kapitel 4.4.7 beschriebene Tooltip eingefügt.

„initStyles“

In diesem Abschnitt wird beschrieben, welche CSS-Styles dynamisch erstellt und hinzugefügt werden müssen, damit ein Diagramm korrekt dargestellt werden kann. Dazu wird die Funktionsweise der Methode „initStyles“ beschrieben.
Innerhalb der Methode werden zuerst die Pixelwerte für die Koordinaten der Datenpunkte berechnet. Anschließend werden die beiden Methoden „stylesDatapoints“ und „stylesAxisLabels“ aufgerufen. Sowohl die HTML-Elemente der Datenpunkte, als auch die HTML-Elemente der Labels einer Achse sind absolut positioniert. Dadurch können die Koordinaten als Pixelwerte über die beiden CSS-Properties „left“ und „bottom“ an die entsprechenden HTML-Elementen übergeben werden. Zusätzlich wird im Rumpf der Methode „stylesDatapoints“ jedem Datenpunkt die Farbe der dazugehörigen Datenserien zugewiesen. Neben dem Ausrichten der Achsen-Labels und der Datenpunkte, werden in der Methode „initStyles“ auch die im Konstruktor geladenen

Paint-Worklets den entsprechenden HTML-Elementen als CSS-„background“ zugewiesen. Damit die Paint-Worklets korrekt aufgerufen werden können, müssen zusätzlich die Properties des jeweiligen Worklets den HTML-Elementen hinzugefügt werden. Für das Hinzufügen der CSS-Properties wird das unter Kapitel 4.1.3 beschriebene Typed Object Model API verwendet.

„initEvents“

Die Methode „initEvents“ wird dafür verwendet, um die Diagramme interaktiv zu gestalten. Dazu werden innerhalb der Methode verschiedene Events registriert, welche auf die Ereignisse von Benutzerinteraktionen reagieren. Im Folgenden werden die drei grundlegenden Events einer Diagramm-Komponente anhand der dazugehörigen Methoden beschrieben.
Im Rumpf der Methode „eventsDatapoint“ wird jedem Datenpunkt-Element ein Eventlistener hinzugefügt. Interagiert der Benutzer mit einem Datenpunkt, werden die Methoden der in Kapitel 4.4.7 beschriebenen „Tooltip“-Komponente aufgerufen, um weitere Informationen ein- oder auszublenden. Zusätzlich wird beim Klick auf einen Datenpunkt eine farbige, vertikale Linie in den Hintergrund des Diagramms gezeichnet, die den entsprechenden Punkt schneidet und hervorhebt. Damit sich nach einer Veränderung des Browser-Viewports alle Inhalte eines Diagramms an den neuen Viewport anpassen, wird in der Methode „eventsResize“ ein „resize“ Eventlistener registriert. Dieser wird benötigt, um beispielsweise die Position der Datenpunkte erneut zu berechnen. Löst ein Ereignis das „resize“ Event aus, wird die Methode „setChartSize“ aufgerufen und die Größe der Zeichenfläche neu bestimmt. Basierend auf den neuen Maßen der Zeichenfläche können die Koordinaten der Datenpunkte erneut bestimmt und somit die Position der HTML-Elemente angepasst werden. Das dritte Event, welches jede Diagramm-Klasse standardmäßig besitzt, wird über die Methode „eventsHeader“, der in Kapitel 4.4.6 beschriebenen Komponente „Header“, registriert.

4.4.4 Liniendiagramm

In diesem Kapitel werden die Eigenschaften eines Liniendiagramms beschrieben und erläutert. Außerdem wird herausgearbeitet, für welchen Kontext sich diese Darstellungsweise besonders eignet. Darüber hinaus werden die Herausforderungen bei der Erstellung eines Liniendiagramms thematisiert. Abschließend werden die technische Umsetzung und die Lösungsansätze der Herausforderungen, basierend auf der unter Kapitel 4.4.3 beschriebenen generischen Diagrammstruktur, erläutert.
Ein Liniendiagramm besteht in der Regel aus einer x- und einer y-Achse, welche orthogonal zueinander stehen. Die Ausnahme bilden dreidimensionale Liniendiagramme,

welche über eine dritte, räumliche Achse verfügen. Der Fokus dieser Ausarbeitung liegt auf der Erstellung eines zweidimensionalen Liniendiagramms, weshalb die räumliche Achse nicht näher beschrieben wird. Jede Achse eines Liniendiagramms beschreibt ein Merkmal, welches häufig neben oder unter der entsprechenden Achse textlich beschrieben wird. Die Datenpunkte eines Liniendiagramms sind über eine Linie miteinander verbunden, dadurch können besonders gut Verläufe von einer oder mehreren Datenreihen dargestellt und miteinander verglichen werden. Aufgrund dessen können mit einem Liniendiagramm einfach Trends oder Veränderungen visualisiert werden.[37]

Herausforderungen

Bei der Erstellung eines Liniendiagramms ist zu beachten, dass sowohl für die x-Achse als auch für die y-Achse eine dynamisch anpassbare Skalierung erstellt werden muss. Darüber hinaus sollte die Skalierung der Rechtsachse, neben Daten in Text- oder Zahlenform, auch Daten unterstützen, die in einem Datumsformat vorliegen. Liegt ein Datumsformat vor, muss dieses gegebenenfalls zusätzlich formatiert werden, um eine korrekte Beschriftung der Skalierung zu gewährleisten. Eine weitere Herausforderung bei der Erstellung eines Liniendiagramms besteht darin, die Position der Datenpunkte und Gitternetzlinien korrekt zu berechnen, damit diese mit der Skalierung übereinstimmen. Außerdem muss eine effiziente Lösung gefunden werden, um die Datenpunkte einer Datenserie mit einer Linie zu verbinden.

[37] [Sta12, 334]

Umsetzung

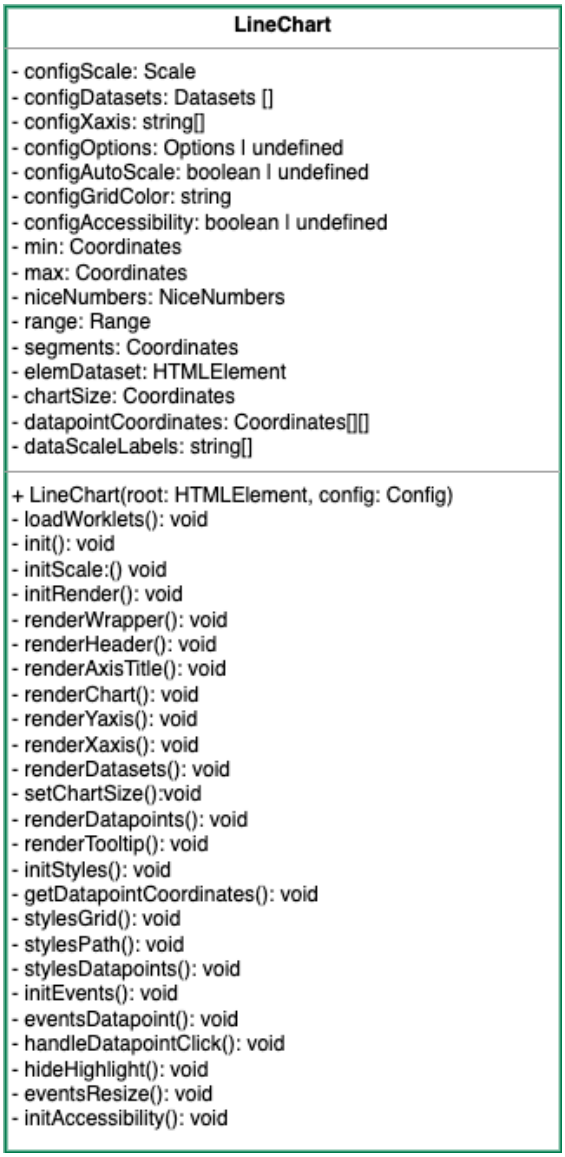


Abbildung 7: Klassendiagramm - LineChart

Aufgrund der Tatsache, dass ein zweidimensionales Liniendiagramm zwei Achsen besitzt, müssen die folgenden Properties, der in Abbildung 7 dargestellten Klasse „LineChart“, um die entsprechenden Werte für die X-Achse erweitert werden:

- „min“
- „max“
- „range“
- „segments“

Auf dieser Basis kann eine allgemeine Skalierung für die x-Achse erstellt werden. Dabei wird die Anzahl der Intervallschritte der Skalierung und die Beschriftung der Skalierung für die x-Achse anhand der vom Benutzer in der Konfiguration angegebenen Labels bestimmt. Im Rumpf der Methode „initScale“ wird der Property „min.x“ standardmäßig die Zahl null und der Property „max.x“ der Wert, der sich aus der Anzahl an Elementen innerhalb des in der Konfiguration angegeben Arrays „xaxis“ ergibt, zugewiesen. Anhand dessen können anschließend die Werte für die Properties „range.x“ und „segments.x“ berechnet werden. „Range.x“ entspricht der Differenz aus „max.x“ und „min.x“, während „segments.x“ dem Wert von „max.x“ gleicht.
Hat der Benutzer in der Konfiguration den Skalierungstyp „date“ angegeben wird am Ende des Rumpfes der Methode „initScale“ die Funktion „getDateScaleLabels“ aufgerufen. Innerhalb der Funktion wird die Anzahl an Tagen zwischen dem ältesten und dem jüngsten Datum der übergebenen Daten berechnet. Basierend auf der Differenz an Tagen wird anschließend in einem Switch-Statement überprüft, ob die Skalierung in Sekunden, Minuten, Stunden, Tagen, Monaten oder Jahren angegeben werden soll. Tritt beispielsweise der Fall ein, dass sich eine Monats-Skalierung am besten eignet, wird die Funktion „generateMonthScale“ aufgerufen. Die Funktion bestimmt zuerst anhand des jüngsten Datums den letzten Tag innerhalb des jeweiligen Monats. Anschließend wird, basierend auf dem ältesten Datum, der erste Tag innerhalb des jeweiligen Monats ermittelt. Danach kann die Gesamtzahl an Tagen und Monaten zwischen diesen beiden Werten ermittelt werden. Für die korrekte Berechnung dieser Datumsangaben wird die JavaScript-Library „Luxon“ verwendet. Die Anzahl an Monaten wird anschließend als Wert für den Schleifenzähler einer For-Schleife verwendet. Dadurch können die Namen und das Jahr aller Monate zwischen dem jüngsten und dem ältesten Datum ermittelt und dem Array „labels“ hinzugefügt werden. Das Array wird daraufhin zusammen mit der Gesamtzahl an Tagen in einem Objekt gespeichert und an die Funktion zurückgeliefert. Zum Schluss werden den Properties „max.x“, „segments.x“ und „dateScaleLabels“ die neuen Werte aus dem zuvor bestimmten Objekt zugewiesen. Danach kann die Skalierung über die Methode „renderXAxis“ in das DOM gerendert werden.
Die Position der Datenpunkte wird mithilfe der Methode „getDatapointCoordinates“ bestimmt. Im Rumpf der Methode wird über alle Datenreihen des Diagramms iteriert. Bei jeder Iteration wird die Funktion „getLinePoints“ aufgerufen. Für die Berechnung der Datenpunkte wird die Höhe und die Breite der Zeichenfläche, als auch die Differenz zwischen dem höchsten und niedrigsten Wert der jeweiligen Achse, benötigt. Mithilfe dieser Angaben können die Pixelwerte für alle Datenpunkte einer Datenserie bestimmt werden. Dazu müssen die x- und y-Koordinaten in Relation zu der Dia-

grammgröße und der Skalierung gesetzt werden. Nachdem die Position der einzelnen Datenpunkte berechnet wurde, werden die Pixelwerte einem neuen Array hinzugefügt und als Rückgabewert an die Funktion übergeben.
Im Anschluss an die Berechnung der Position der Datenpunkte, kann die Linie zum Verbinden der einzelnen Punkte gezeichnet werden. Für das Zeichnen der Linie wird ein Paint-Worklet verwendet. Das Worklet „path-line“ besitzt die beiden Properties „--path-points“ und „--path-color“. Mithilfe der Property „--path-color“ wird zuerst die Farbe der zu zeichnenden Linie festgelegt. Die Property „--path-points“ enthält die Koordinaten der einzelnen Datenpunkte. Diese werden im nächsten Schritt an die „CanvasRenderingContext2D.lineTo“-Methode übergeben. Die Methode ermöglicht es auf einer Zeichenfläche eine Linie zu einem Punkt zu malen. Um eine Linie zu zeichnen, die alle Datenpunkte miteinander verbindet, müssen dementsprechend die Koordinaten der einzelnen Datenpunkte innerhalb einer For-Schleife an die Methode übergeben werden.
Die Gitternetzlinien eines Liniendiagramms werden ebenfalls mithilfe eines Paint-Worklets erstellt. Die Farbe der Linien wird über die Property „--grid-color“ bestimmt. Über die beiden Properties „segmentsX“ und „segmentsY“ des Worklets „grid-line“ kann definiert werden, wie viele horizontale und vertikale Linien für das Gitternetz gezeichnet werden sollen. Um die Größe eines Segments zu bestimmen, muss zuerst die Breite der Zeichenfläche durch den Wert von „segmentsX“ und die Höhe der Zeichenfläche durch den Wert von „segmentsY“ dividiert werden. Anschließend können die vertikalen und horizontalen Linien jeweils in einer For-Schleife gezeichnet werden. Dabei wird nach jeder Iteration der Wert des Schleifenzählers um die Höhe, beziehungsweise um die Breite eines Segments vergrößert. Mithilfe der letzten Property „--grid-highlight“ des Worklets kann zusätzlich eine Stelle im Diagramm hervorgehoben werden, indem an den entsprechenden Koordinaten eine gestrichelte, dunkelgraue, vertikale Linie gezeichnet wird. Zum Zeichnen und Positionieren der Linien wird erneut die „CanvasRenderingContext2D.lineTo“- und „CanvasRenderingContext2D.moveTo“-Methode verwendet.

4.4.5 Radardiagramm

Ein Radardiagramm ist ein Diagramm zum Darstellen von Datenpunkten und deren Variationen.

> *Es kann mehrere gleichwerte Kategorien wiedergeben, die jeweils über eine eigene Achse verfügen, an der die jeweiligen Werte übertragen werden.*[38]

Die Achsen eines Radardiagramms sind dabei kreisförmig angeordnet. Aufgrund der kreisförmigen Anordnung der Achsen und der Tatsache, dass bei einem Radardiagramm die einzelnen Datenpunkte einer Datenreihe miteinander verbunden sind, müssen mindesten drei Kategorien bei der Initialisierung eines Diagramms angegeben werden. Andernfalls liegen die Linien aufeinander, wodurch das Diagramm nicht mehr aussagekräftig ist. Aus diesem Grund bestehen in der Regel die meisten Radardiagramme aus fünf bis sieben Kategorien. Abhängig von der Anzahl an Achsen bilden die Gitternetzlinien eines Radardiagramms beispielsweise die Form eins Penta-, Hexa-, oder Heptagons. Ein Radardiagramm eignet sich besonders, wenn die Punkte von zwei oder mehreren verschiedenen Datenreihen miteinander verglichen werden sollen.
Im Folgenden wird die in Abbildung 8 dargestellten Komponente „RadarChart“ beschrieben. Dafür werden die Herausforderungen bei der Erstellung eines Radardiagramms aufgezeigt und darauf aufbauend die technische Umsetzung zur Lösung der Herausforderungen beschrieben. Der generelle Aufbau der Diagramm Komponente wird dabei nicht erläutert, da dieser bereits in Kapitel 4.4.3 skizziert wird.

Herausforderungen

Die Problematik bei der Erstellung eines Radardiagramms besteht darin, dass die Anzahl an Achsen, im Vergleich zu einem Liniendiagramm, nicht statisch sondern dynamisch ist. Zusätzlich verändert sich die Position einer Achse im Kreis abhängig von der Gesamtzahl an Kategorien, die ein Diagramm besitzt. Dadurch muss das Layout und somit auch die Gitternetzlinien eines Radardiagramms immer an die Anzahl der Kategorien angepasst werden. Darüber hinaus müssen neben den Achsen auch die Position der Kategorie-Labels und die Position der Datenpunkte dynamisch berechnet werden. Eine weitere Herausforderung besteht darin, die Datenpunkte der einzelnen Datenreihen miteinander zu verbinden und die daraus entstehende Fläche gegebenenfalls zu füllen.

[38] [Sta12, 358]

Umsetzung

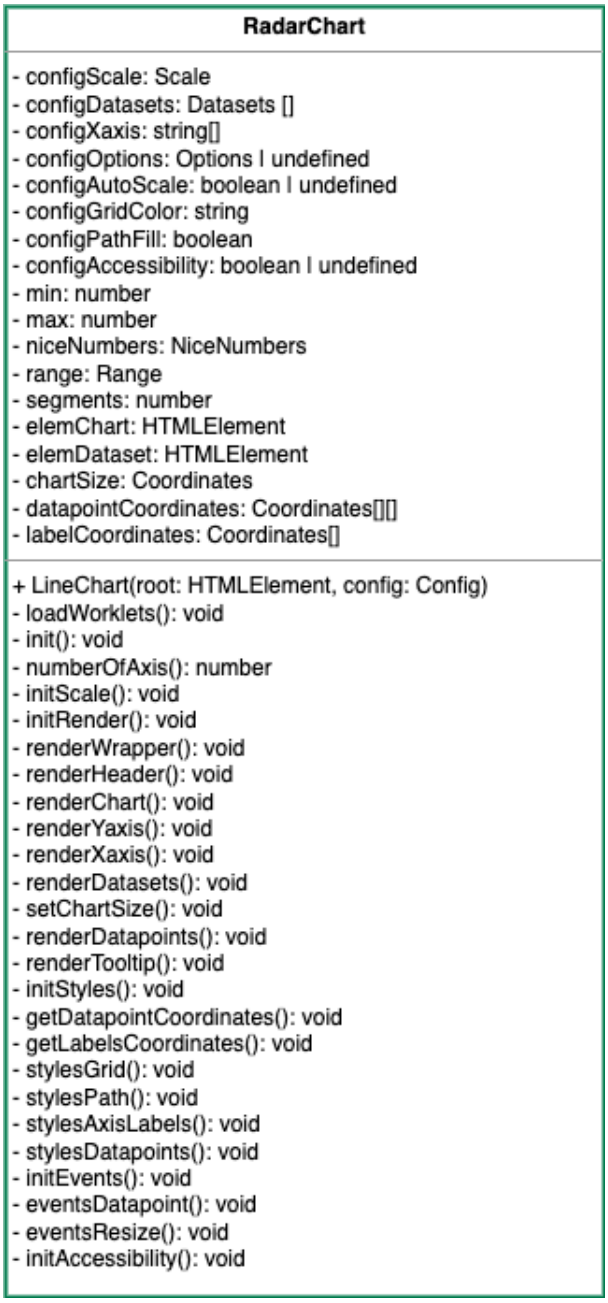


Abbildung 8: Klassendiagramm - RadarChart

Für die Erstellung eines Radardiagramms muss zuerst die Anzahl an Achsen bestimmt werden. Diese wird anhand der Einträge innerhalb des in der Konfiguration angegebenen Arrays „xaxis“ festgelegt. Mithilfe der Anzahl an Achsen können anschließend die Koordinaten der einzelnen Datenpunkte berechnet werden. Dazu wird die Methode „getDatapointCoordinates“, der in Abbildung 8 dargestellten Klasse „RadarChart“, aufgerufen. Im Rumpf der Methode wird über alle Datenserien eines Diagramms ite-

riert. Dabei werden die Werte einer Datenserie als Array, die Größe des Diagramms in Pixeln und die Differenz zwischen dem höchsten und niedrigsten Wert an die Helfer-Funktion „getRadarPoints“ übergeben. Innerhalb der „getRadarPoints“-Funktion wird zu Beginn der Mittelpunkt des Diagramms und das Winkelmaß zwischen den Achsen berechnet. Das Winkelmaß lässt sich mit folgender Formel berechnen:

$$Winkelmaß = \pi * 2/Achsenanzahl \tag{4.3}$$

Anschließend wird mit einer For-Schleife über alle Werte einer Datenserie iteriert. Bei jedem Durchlauf wird zuerst der Radius des Kreises, anhand dessen die Datenpunkte ausgerichtet werden sollen, bestimmt. Danach wird die Größe des Kreises und der Wert des Schleifenzählers an die Funktion „getPolygonPos“ übergeben. Die Funktion „getPolygonPos“ errechnet die x- und y-Position eines Datenpunktes, indem der Radius des zuvor errechneten Kreises mit dem Sinus oder Kosinus des Produktes aus Achse und Winkelmaß multipliziert wird.

$$x = r * sin(Achse * Winkel) \tag{4.4}$$

$$y = r * cos(Achse * Winkel) \tag{4.5}$$

Abschließend werden die errechneten Pixelwerte an die Funktion zurückgegeben und einem Array hinzugefügt. Das Array mit den Pixelwerten der Datenpunkte wird am Ende an die Methode „getDatapointsCoordinates“ zurückgeliefert.
Die Position der Kategorie-Labels wird mithilfe der Methode „getLabelsCoordinates“ bestimmt. Die Methode beinhaltet ebenfalls den Aufruf der Funktion „getRadarPoints“. Anstelle der Datenserien wird allerdings ein Array übergeben, dessen Menge an Elementen der Anzahl an Achsen gleicht. Der Wert jedes Elementes entspricht dem höchsten Wert der y-Achsen Skalierung. Diesem höchsten Wert wird ein zusätzlicher Abstand hinzugefügt, damit sich die Labels der Kategorien außerhalb des Diagramms befinden und keine Inhalte überdecken.
Nachdem die Position der Datenpunkte und Kategorie-Labels bestimmt wurde, können die Methoden „stylesAxisLabels“ und „stylesDatapoints“ der Klasse „RadarChart“ aufgerufen werden. Innerhalb der beiden Methoden werden den HTML-Elementen der Datenpunkte und der Kategorie-Labels die korrekte Position zugewiesen, indem die CSS Properties „left“ und „bottom“ mit den entsprechenden Pixelwerten an die HTML-Elemente übergeben werden.
Die Herausforderung, die einzelnen Datenpunkte einer Datenserie miteinander zu verbinden, wird durch ein Paint-Worklet gelöst. Das Worklet „path-radar“ besitzt drei Properties „--path-points“, „--path-fill“ und „--path-color“. Diese werden als CSS-

Variablen beim Aufruf der Methode „stylesPath“ an das entsprechende HTML-Element einer Datenserie übergeben. Die Property „--path-points“ ist ein Array mit den Koordinaten der zuvor berechneten Datenpunkte. Zuerst wird innerhalb des Worklets festgelegt, welche Farbe die zu zeichnende Linie hat und wie breit diese ist. Anschließend wird über die Datenpunkte iteriert und mithilfe der „CanvasRenderingContext2D.lineTo“-Methode eine Linie zu den Koordinaten des nächsten Datenpunktes gezeichnet. Dabei muss, wie in Kapitel 4.1.1 beschrieben, darauf geachtet werden, dass das Vorzeichen der y-Koordinate umgekehrt wird. Abschließend wird überprüft, ob die Property „--path-fill“ dem Wert „true“ entspricht. Ist dies der Fall, wird mittels der „CanvasRenderingContext2D.fill“-Methode die gezeichnete Fläche der Linien mit Farbe gefüllt.

Um die Gitternetzlinien eines Radardiagramms dynamisch zeichnen zu können, wird ebenfalls ein Paint-Worklet verwendet. Die Werte für die Properties des Worklets „grid-radar“ werden im Rumpf der Methode „stylesGrid“ der „RadarChart“-Klasse zugewiesen. Das Worklet besitzt die Properties „--grid-xaxis“, „--grid-segments“ und „--grid-color“. Innerhalb des Worklets wird zuerst in einer For-Schleife die Größe des Radius von jedem Segment berechnet. Dazu wird die Hälfte der Zeichenfläche durch die Anzahl der Intervallschritte in der y-Achsen Skalierung dividiert. Anschließend werden die errechneten Radien einem Array hinzugefügt. Die Elemente des Arrays werden danach in einer For-Schleife an die Funktion „getPolygonPos“ übergeben. Die Funktion „getPolygonPos“ bestimmt die Koordinaten der Eckpunkte eines Polygons, wie bereits zuvor in diesem Kapitel beschrieben. Infolgedessen werden die Gitternetzlinien mithilfe der „CanvasRenderingContext2D.lineTo“-Methode anhand der Koordinaten der Eckpunkte der einzelnen Segmente gezeichnet. Um die Achsen des Diagramms abzubilden, werden aus der Mitte der Zeichenfläche Linien zu den Eckpunkten des Segments mit dem größten Radius gezeichnet.

4.4.6 Header

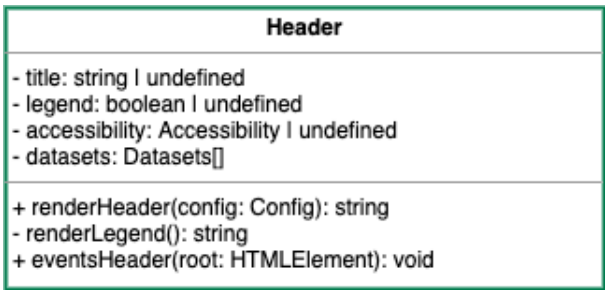


Abbildung 9: Klassendiagramm - Header

Das in Abbildung 9 dargestellte Klassendiagramm zeigt den Aufbau der „Header"-Komponente. Die Komponente stellt den Kopfbereich eines Diagramms dar und besteht aus einem Diagrammtitel und einer Legende. Aufgerufen wird die Komponente innerhalb der „initRender"-Methode einer Diagramm-Klasse. Im Folgenden wird die Funktionsweise der Komponente erläutert und auf die Methoden und Attribute der Klasse eingegangen.
Die öffentliche Methode „renderHeader" hat einen Parameter vom Typ „Config" und liefert als Rückgabewert die HTML-Elemente der Komponente in Form eines Strings. Beim Aufruf der „renderHeader"-Methode werden zuerst die Properties „title", „legend", „accessibility" und „datasets" mit den Werten des Parameters „config" initialisiert. Anschließend wird ein Template-String mit der grundlegenden Struktur der Komponente gebildet und an die Funktion zurückgegeben. Innerhalb des Template-Strings wird überprüft, ob die Property „legend" dem Wert „true" entspricht. Trifft dies zu, wird die Methode „renderLegend" aufgerufen. Anderenfalls wird die Methode nicht aufgerufen.
Die „renderLegend"-Methode liefert die dynamisch generierten HTML-Elemente der Legende in Form eines Strings an die Funktion „renderHeader" zurück. Für die Erstellung der Legende werden die Informationen zu den einzelnen Datensätzen des Diagramms benötigt. Dazu wird über das Array der Property „dataset" iteriert. Dabei wird für jedes Element des Arrays ein HTML „<button>" Element erstellt mit dem Namen und der Farbe des jeweiligen Datensatzes. Ist die Property „accessibility" gleich „true" werden die erstellten HTML „<button>" Elemente zusätzlich mit einem aussagekräftigen „aria-label"-Attribut versehen, um die Komponente barrierefrei zu gestalten. Abschließend kann die „Header"-Komponente in den DOM gerendert werden.
Damit die einzelnen Datensätze eines Diagramms über die Legende ein- und ausgeblendet werden können, muss die öffentliche Methode „eventsHeader" in der „init-

Events“-Funktion einer Diagramm-Komponente aufgerufen werden. Die „eventsHeader“-Methode hat einen Parameter des Typs „HTMLElement“. Mithilfe des Parameters müssen zuerst alle „<button>“ Elemente innerhalb der Legende abgefragt und in einer Variablen gespeichert werden. Anschließend erhält jedes der Elemente einen Eventlistener, der durch einen Klick auf das entsprechende Element den dazugehörigen Datensatz ein- oder ausblendet.

4.4.7 Tooltip

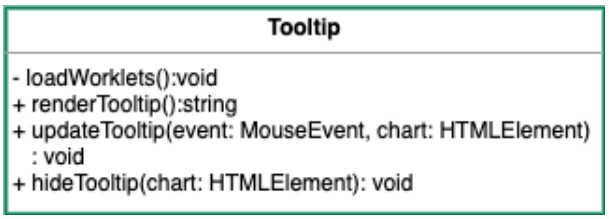


Abbildung 10: Klassendiagramm - Tooltip

Die „Tooltip“-Komponente zeigt bei der Interaktion mit einem Datenpunkt ein Fenster an, welches nähere Informationen zu dem ausgewählten Punkt enthält. Beispielsweise können das die exakten Koordinaten des Datenpunktes oder der Name des Datensatzes sein, zu dem der Datenpunkt gehört. Die Herausforderung bei der Erstellung einer „Tooltip“-Komponente besteht darin, dass die Informationen der Komponente, basierend auf einer Nutzerinteraktion, dynamisch angepasst werden müssen. Im Folgenden wird die Funktionsweise der „Tooltip“-Komponente anhand des in Abbildung 10 zu sehenden Klassendiagramms erläutert.
Bei der Initialisierung der „Tooltip“-Komponente wird zuerst die Methode „loadWorklets“ im Konstruktor der Klasse aufgerufen. Daraufhin wird das Paint-Worklet mit dem Namen „tooltipWorklet“ geladen. Anschließend kann das Worklet als CSS-„background-image“ verwendet werden. Mithilfe des Worklets werden nicht mehrere Pseudoelemente benötigt um ein Dialogfenster mit Pfeil darzustellen, sondern nur ein „background-image“.
Über die öffentliche Methode „renderTooltip“ wird ein Template-String mit der grundlegenden Struktur des Tooltips erstellt. Anschließend wird der String als Rückgabewert an die Methode zurückgeliefert und kann somit an die entsprechende Stelle innerhalb einer Diagramm-Komponente gerendert werden.
Damit die „Tooltip“-Komponente mit den entsprechenden Informationen eines Datenpunktes befüllt wird, muss zu jedem Datenpunkt ein Eventlistener hinzugefügt werden, der die Methode „updateTooltip“ ausführt. Die Methode besitzt zwei Parameter. Der erste Parameter ist vom Typ „MouseEvent“ und der zweite vom Typ „HTML-

Element“. Anhand der Informationen des „MouseEvents“ kann festgestellt werden, mit welchem Datenpunkt der Nutzer interagiert hat. Dadurch kann die Position des Tooltips bestimmt und die Information, zu welchem Datensatz der Punkt gehört, abgefragt werden. Abschließend kann, mithilfe der gesammelten Informationen, die „Tooltip“-Klasse neu gerendert werden.
Soll die „Tooltip“-Komponente wieder ausgeblendet werden, muss die öffentliche Methode „hideTooltip“ aufgerufen werden. Im Rumpf der Methode wird das oberste HTML-Element des Tooltips ermittelt und über eine CSS-Klasse ausgeblendet.

4.4.8 Accessibility

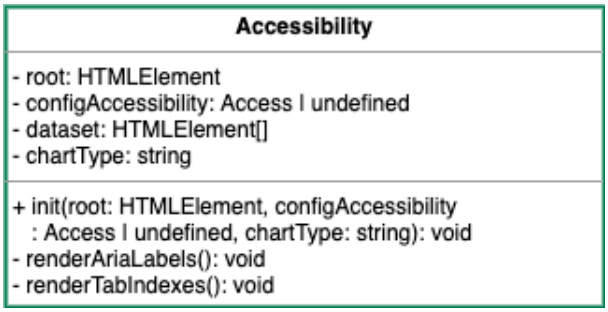


Abbildung 11: Klassendiagramm - Accessibility

Das Klassendiagramm in Abbildung 11 zeigt die „Accessibility“-Komponente. Wählt ein Benutzer in der Konfiguration eines Diagramms das Feature „Accessibility“ aus, wird die Komponente über die Methode „initAccessibility“ aufgerufen. Mithilfe der Komponente werden die Inhalte eines Diagramms zugänglich für Screenreader gestaltet. Darüber hinaus werden die Diagramme für die Tastatursteuerung optimiert. Im Folgenden wird die Klasse näher beschrieben und erläutert.
Die „Accessibility“-Komponente wird über ihre öffentliche „init“-Methode initialisiert. Damit die „init“-Methode aufgerufen werden kann, benötigt sie die Attribute „root“, „configAccessibility“ und „chartType“. Die Werte der Attribute werden im Rumpf der Methode den gleichnamigen Properties zugewiesen. Zusätzlich wird die Property „datasets“ initialisiert und die beiden Methoden „renderAriaLabels“ und „renderTabIndexes“ aufgerufen.
Innerhalb der Methode „renderAriaLabels“ wird über alle Datensätze und Datenpunkte eines Diagramms iteriert. Dabei wird jedem Element ein „aria-label“-Attribut zugewiesen. Der Wert des Attributs entspricht einem Template-String, welcher Informationen zu den einzelnen Datensätzen und Datenpunkten enthält. Dazu zählt zum Beispiel die Anzahl der Datenpunkte eines Datensatzes oder die genauen Koordinaten eines Datenpunktes. Zusätzlich wird in der Methode „renderAriaLabels“ überprüft, ob eine

allgemeine Beschreibung für den Inhalt des Diagramms angegeben wurde. Ist dies der Fall, wird die Beschreibung dem entsprechenden HTML-Element zugewiesen. Anderenfalls wird das Diagramm ohne Inhaltsbeschreibung gerendert.
Damit die Inhalte eines Diagramms besser zugänglich für die Navigation mit der Tastatur sind, wird in der Methode „renderTabindex“ jedem Datenpunkt das Attribut „tabindex“ mit dem Wert „0“ hinzugefügt. Der Wert „0“ legt fest, dass die Datenpunkte in der Reihenfolge, in der sie in das DOM gerendert wurden, fokussierbar sind.

5 Evaluierung

In diesem Kapitel wird die aus der Entwicklung hervorgegangene Library „HoudiniCharts“ evaluiert. Dabei werden die Diagramm-Komponenten der Library anhand ihrer Performance, Interaktivität und Barrierefreiheit beurteilt. Als Referenz für die Evaluation werden die beiden am häufigsten von der Internet-Plattform „npm“ heruntergeladenen Diagramm-Libraries „Chart.js“ und „Highcharts“ herangezogen.[39] Neben dem Faktor Popularität eigenen sich die beiden Libraries außerdem optimal für den Vergleich, da „Chart.js“ Canvas- und „Highcharts“ SVG-basiert ist. Aufgrund dessen werden die beiden aus der Analyse hervorgegangenen Technologien zur Datenvisualisierung im Web (siehe Kapitel 2.1), bei der Gegenüberstellung mit der entwickelten Library „HoudiniCharts“ berücksichtigt.
Im Folgenden wird die Testumgebung beschrieben, welche die Grundlage für die Beurteilung darstellt. Aufbauend auf der Testumgebung wird anschließend die Evaluation durchgeführt. Dabei ist festzuhalten, dass die im Rahmen dieser Ausarbeitung definierten Anforderungen vollständig erfüllt wurden. Die Evaluation ist in die Unterkapitel Performance, Interaktivität und Barrierefreiheit unterteilt. In diesen Unterkapiteln werden die Ergebnisse der Evaluation analysiert und ausgewertet.

5.1 Testumgebung

Als Basis für die Durchführung der Evaluation muss eine normierte Testumgebung geschaffen werden, damit eine Vergleichbarkeit zwischen den Diagrammen der einzelnen Anbietern gewährleistet werden kann. Aufgrund der Tatsache, dass für die Entwicklung der Library „HoudiniCharts“ Schnittstellen von CSS Houdini verwendet werden, deren Kompatibilität auf Chromium-basierte Browser reduziert ist (siehe Abbildung 2), wird für die Evaluation ausschließlich der Browser Google Chrome verwendet. Darüber hinaus wird die entwickelte Webanwendung um Beispiele der Libraries „Chart.js“ und „Highcharts“ erweitert.

[39] [Joh]

Abbildung 12: Screenshot - Webanwendung

Der Aufbau der Webanwendung wird in Abbildung 12 visualisiert. Über die Startseite der Webanwendung hat der Nutzer die Möglichkeit zu den einzelnen Diagrammen der verschiedenen Anbieter zu navigieren. Insgesamt werden zu jedem Anbieter drei Linien- und drei Radardiagramme erstellt. Diese sind in die Kategorien „Klein“, „Groß“ und „A11y“ unterteilt. Die Kategorien „Klein“ und „Groß“ unterscheiden sich in der Anzahl an darzustellenden Daten. In der Kategorie „Klein“ werden die Diagramme mit einer geringen Anzahl und in der Kategorie „Groß“ werden die Diagramme mit einer Vielzahl an Datensätzen initialisiert. Die Kategorie „A11y“ soll eine barrierefreie Version des jeweiligen Diagrammtyps abbilden. Mit Ausnahme der barrierefreien Version, werden die Daten der Diagramme über die folgenden API-Endpoints bezogen:

Liniendiagramm

Kategorie	Daten	API-Endpoints
Klein	7	`https://api.corona-zahlen.org/germany/history/cases/7`
Groß	614	`https://api.corona-zahlen.org/germany/history/cases`

Radardiagramm

Kategorie	Daten	API-Endpoints
Klein	12	`https://api.corona-zahlen.org/germany/age-groups`
Groß	160	`https://api.corona-zahlen.org/states/age-groups`

Tabelle 3: Datensätze - Testumgebung

Die in Tabelle 3 aufgelisteten API-Endpoints der „Robert Koch-Institut API“ liefern die aktuellen Corona-Fallzahlen in Deutschland. Weitere Informationen können in der Dokumentation der Rest-API, welche unter dem Link `https://api.corona-zahlen.org/docs/` zu finden ist, abgerufen werden.[40] Damit die Daten des Rest-API verwendet werden können wird ein Adapter benötigt, um die in einem JavaScript Object Notation (JSON)-Format gelieferten Daten zu formatieren. Die formatierten Daten können anschließend an die Konfigurations-Objekte der jeweiligen Libraries

[40] [Lü]

übergeben werden. Für die Liniendiagramme in der Kategorie „Klein“ werden die aktuellen Corona-Infektionszahlen in Deutschland, innerhalb der letzten sieben Tage, von der API abgefragt. Bei der Kategorie „Groß“ werden hingegen alle Daten vom Beginn der Messung, bis hin zum aktuellen Datum angefordert. Zum Durchführungszeitpunkt der Messungen enthält die Datenkategorie „Groß“ 614 Einträge. Damit die Daten auch in einem Radardiagramm sinnvoll dargestellt werden können, wird der API-Endpoint abgefragt, welcher die Daten aller Corona-Infektionszahlen in Deutschland, unterteilt nach Geschlecht und Altersgruppen, liefert. Für die Variante des Radardiagramms mit einer großen Datenmenge werden die Daten zusätzlich in die einzelnen Bundesländer von Deutschland unterteilt. Dadurch müssen die Diagramme in dieser Kategorie 160 Daten visualisieren.
Jedes der Diagramme soll einen Diagrammtitel, eine Diagrammlegende und eine Achsenbeschriftung besitzen. Außerdem sollen alle Animationen für die Diagramme deaktiviert werden, damit die Struktur und das Verhalten der Diagramm-Komponenten möglichst einheitlich sind. Dazu werden die Konfigurations-Objekte der jeweiligen Komponenten angepasst. Zusätzlich werden für die Diagramme standardisierte HTML-Templates entwickelt, in welche die Komponenten gerendert werden. Die Größen und die Seitenverhältnisse der verschiedenen Diagrammtypen werden ebenfalls einheitlich festgelegt.

5.2 Performance

Für die Beurteilung der Performance der Libraries „HoudiniCharts“, „Charts.js“ und „Highcharts“ wurde ein eigenes Messverfahren entwickelt, welches die Ausführungszeit des Programmiercodes, bis ein Diagramm vollständig dargestellt wird, berechnet. Einleitend wird der Aufbau und die Struktur des Testverfahrens beschrieben. Im Anschluss wird die Notwendigkeit einer programmatischen Lösung zur Durchführung der Tests erläutert. Anschließend wird anhand eines Schaubildes die Funktionalität der entwickelten Anwendung zur Umsetzung der Performancemessungen dargestellt. Danach wird die Verfahrensweise zur Auswertung und Gegenüberstellung der Ergebnisse beschrieben. Abschließend werden die Ergebnisse der Messungen tabellarisch dargestellt, analysiert und miteinander verglichen.

5.2.1 Testverfahren

Das Ziel des Testverfahrens besteht darin, die Ausführungszeit des Programmiercodes der Libraries „HoudiniCharts“, „Charts.js“ und „Highcharts“ unter verschiedenen Rahmenbedingungen zu messen. Anhand der Messergebnisse wird anschließend die Performance der entwickelten Library „HoudiniCharts“ evaluiert.
Für die Messung der Ausführungszeit wird die JavaScript-Methode „performance.now“ verwendet. Die „performance.now“-Methode gibt einen Zeitwert in Millisekunden zurück, welcher den Zeitpunkt, an dem die Methode aufgerufen wurde, markiert. Wird die Methode vor und nach der Initialisierung einer Diagramm-Komponente aufgerufen kann, anhand der Differenz der beiden Werte, die Ausführungszeit des Programmiercodes bestimmt werden. Damit die Ausführungszeit ausgewertet werden kann, wird der errechnete Wert anschließend der Property „window.myData.performance“ zugewiesen.
Damit ein realistischer und repräsentativer Durchschnittswert für die Ausführungszeit des Programmiercodes angegeben werden kann, wird jede Diagramm-Komponente 250 Performancemessungen unterzogen. Die Linien- und Radardiagramme der verschiedenen Anbieter werden außerdem einmal mit einer geringen und einmal mit einer großen Menge an Daten gemessen, um die Ausführungszeit in Abhängigkeit von der Anzahl an darzustellenden Daten zu bestimmen. Zusätzlich werden alle Messungen einmal mit CPU- und Netzwerk-Throtteling, sowie einmal ohne Throtteling durchgeführt. Mithilfe des Throttelings kann simuliert werden, wie sich die Ausführungszeiten auf einem mittelklassigen Smartphone verändern. In Summe werden somit acht unterschiedliche Testläufe für jede Library durchgeführt. Dies entspricht einer Gesamtzahl von 2.000 Messungen pro Library.
Aufgrund der Vielzahl an Messungen und dem begrenzten zeitlichen Rahmen der Ausarbeitung, ist eine manuelle Durchführung und Erfassung der Messergebnisse nicht möglich. Aus diesem Grund muss ein automatisiertes Testverfahren entwickelt werden, dass mehrere Messungen nacheinander durchführen und die Ergebnisse in einer JSON-Datei speichern kann. Dazu wird eine Node.js basierte Anwendung entwickelt, mit deren Hilfe die Performancemessungen programmatisch durchgeführt werden können. Die Grundlage für die Anwendung bildet das Node module Lighthouse. Lighthouse ist ein von Google entwickeltes Tool, um die Qualität von Webseiten, anhand der Kategorien Performance, Search Engine Optimization (SEO), Accessibility, Best practices und Progressive Web App (PWA), zu beurteilen.[41] Die Ergebnisse vieler Audits sind in der Kategorie Performance stark abhängig von der Dateigröße der herunterzuladenden Daten, wodurch die beiden Referenz-Libraries, aufgrund ihres größeren Funktionsum-

[41] [Goob]

fanges und somit höheren Dateigröße, benachteiligt wären. Um die Voraussetzungen objektiv zu gestalten, wird zusätzlich zu den bereits vordefinierten Audits, ein eigenes Audit für die Messung der Ausführungszeit erstellt.

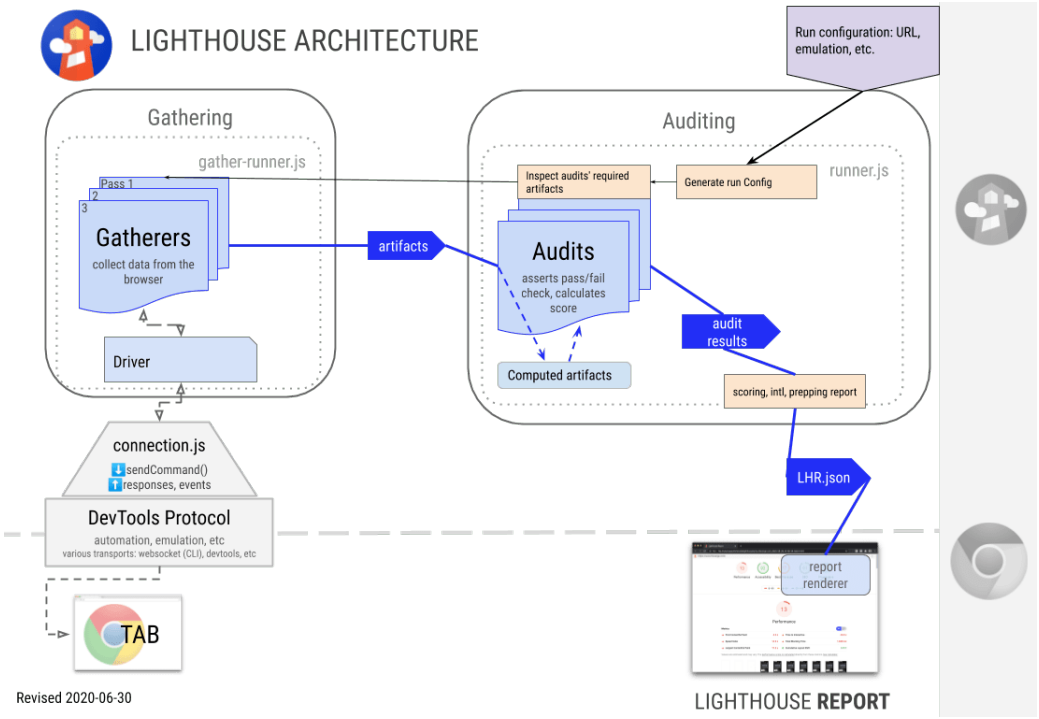


Abbildung 13: Lighthouse - Architektur [Gooa]

Die Architektur von Lighthouse wird in Abbildung 13 dargestellt. Anhand der Architektur wird die Funktionsweise der entwickelten Anwendung zur Performancemessung erläutert. Die Anwendung wird gestartet, indem der nachfolgende Shell-Befehl ausgeführt wird.

```
1    node index.js --url <url-adresse>
```

Listing 5.1: Shell-Befehl Performancemessung

Innerhalb der Anwendung wird zuerst überprüft, ob in dem Verzeichnis „results“ bereits eine Datei mit dem Namen des angegebenen URLs existiert. Ist dies der Fall, wird die Funktion „launchChromeAndRunLighthouse“ aufgerufen. Anderenfalls wird zuerst eine neue JSON-Datei mit den beiden Properties „page“ und „data“ erstellt und danach die Funktion „launchChromeAndRunLighthouse“ aufgerufen. Die Property „page“ wird mit der URL-Adresse der getesteten Seite initialisiert und der Property „data“ wird ein leeres Array zugewiesen. In das Array werden die Ergebnisse der einzelnen Testdurchläufe eingefügt. Im Rumpf der Funktion „launchChromeAndRunLighthouse“ wird zu Beginn eine neue Google-Chrome Instanz gebildet und ein Lighthouse Audit gestartet. Für das Audit werden die URL-Adresse der zu beurteilenden Webseite und

ein Konfigurations-Objekt, welches die Rahmenbedingungen des Audits festlegt, benötigt. Innerhalb des Konfigurations-Objekts kann beispielsweise festgelegt werden, ob ein mittelklassiges Smartphone emuliert werden soll oder nicht. Darüber hinaus kann bestimmt werden, welche Audits durchgeführt werden sollen. In Bezug auf die Beurteilung und Messbarkeit der Performance der herangezogenen Libraries, werden die beiden Audits „dom-size“ und „custom-audit“ gemessen. Das „custom-audit“ gibt die Ausführungszeit des Programmiercodes einer Diagramm-Komponente in Millisekunden an. Damit die gewünschten Audits gemessen werden können, müssen zuerst, im sogenannten „Gathering“, die benötigten Informationen für die Audits gesammelt werden. Für das „custom-audit“ fragt der in der Datei „custom-gatherer.js“ definierte Gatherer, den Wert der Property „window.myData.performance“ ab. Anschließend werden die im „Gathering“ gesammelten Artefakte an das „Auditing“ übergeben. Innerhalb des „Auditings“ wird überprüft, ob ein Audit, basierend auf den gesammelten Informationen, erfolgreich war oder nicht und welcher Wert als Ergebnis für das Audit angezeigt werden soll. Danach werden die Ergebnisse des „Auditing“ als Promise an die Funktion „launchChromeAndRunLighthouse“ zurückgeliefert und die zu Beginn geöffnete Chrome-Instanz geschlossen. Zum Schluss werden die Ergebnisse in einem Objekt gespeichert und mithilfe der Node-Funktion „writeFile“, der zuvor erstellten oder bereits bestehenden JSON-Datei, hinzugefügt.
Mehrere Performancemessung derselben Diagramm-Komponente können anschließend über den folgenden Shell-Befehl ausgeführt werden.

```
1    node multiple-tests.js --runs <anzahl-tests> --url <url-adresse>
```

Listing 5.2: Shell-Befehl Performancemessung, mehrere Durchläufe

Bei dem Aufruf des Befehls wird das Script „multiple-tests.js“ ausgeführt. Innerhalb des Scripts wird in einer Do-While-Schleife die Node-Funktion „execSync“ aufgerufen. Die „execSync“-Funktion besitzt einen Parameter und ermöglicht es, aus einer Node-Anwendung heraus, Shell-Befehle auszuführen. Als Funktionsargument wird der Shell-Befehl zur Ausführung der Performancemessung übergeben (siehe Listing 5.1). Die Anzahl an Iterationen der Do-While-Schleife wird über den Wert des übergebenen Parameters „--runs“ bestimmt.

5.2.2 Ergebnisse

Die Ergebnisse der insgesamt 6.000 durchgeführten Performancemessungen werden mithilfe des Scripts „calcResult.js“ ausgewertet. Das Script wird über den folgenden Befehl ausgeführt.

```
1    node calcResult.js --path <pfad-zu-ergebnis-verzeichnis>
```

Listing 5.3: Shell-Befehl Auswertung Ergebnisse

Bei der Ausführung des Scripts werden zuerst die JSON-Dateien mit den Ergebnissen der getesteten Diagramme geladen und jeweils in einer Variablen gespeichert. Anschließend wird jede der Variablen als Argument an die Funktionen „calcLowest“, „calcAverage“ und „calcHighest“ übergeben. Die Funktionen ermitteln anschließend die minimale, durchschnittliche und maximale Ausführungszeit der Diagramm-Komponenten. Danach werden die Werte in einer Variablen mit dem Namen der jeweiligen Library gespeichert. Zusätzlich wird in der Variablen die Größe des DOMs jeder Diagramm-Komponente festgehalten. Zum Schluss werden die ausgewerteten Ergebnisse aller Libraries in einer neuen JSON-Datei gespeichert.
Der Pfad der neu generierten JSON-Datei kann anschließend als Argument an den folgenden Shell-Befehl übergeben werden.

```
1    node calcDifference.js --path <pfad-zu-result.json>
```

Listing 5.4: Shell-Befehl Vergleich Ergebnisse

Nach dem Aufruf des Shell-Befehls errechnet das Script „calcDifference.js“ den prozentualen Unterschied zwischen den Ausführungszeiten der Diagramm-Komponenten der Library „HoudiniCharts“ und den beiden Referenz-Libraries. Die Ergebnisse werden abschließend in einer neuen JSON-Datei zusammengefasst und gespeichert.
Bei der Beurteilung und dem Vergleich der Ergebnisse ist zu beachten, dass die Ergebnisse der Performancemessungen abhängig von der Performance des jeweiligen Gerätes sind, auf dem die Tests durchgeführt werden. Insbesondere das CPU-Throtteling zum Emulieren eines mittelklassigen Smartphones ist von der Performance des Testgerätes abhängig. Für die Durchführung aller Performancemessungen wurde daher immer dasselbe Testgerät verwendet. Nachfolgend sind die Spezifikationen des verwendeten Testgerätes aufgelistet.

Testgerät	MacBook Air (M1, 2020)
Betriebssystem	macOS Monterey (12.0.1)
Prozessor	Apple M1
Arbeitsspeicher	16gb

Tabelle 4: Spezifikationen - Testgerät

Zusätzlich ist zu beachten, dass bei den Performancemessungen der Diagramm-Komponenten der Library „HoudiniCharts“ nicht die Ausführungszeiten der Worklets inkludiert sind. Dies liegt zum einen daran, dass die Worklets asynchron aufgerufen werden und zum anderen an der Tatsache, dass innerhalb des Kontextes eines Paint-Worklet nicht auf den globalen Kontext einer Webanwendung zugegriffen werden kann.
Aufgrund dieser Eigenschaften ist es schwierig ein Messverfahren zu entwickeln, welches sowohl den genauen Zeitpunkt indem ein Worklet aufgerufen wird feststellt, als auch misst, wie lange die Ausführungszeit des jeweiligen Worklets ist und zusätzlich berechnet, inwiefern diese Zeit die Gesamtausführungszeit der Diagramm-Komponente erhöht.
Um dennoch Richtwerte für die Ausführungszeiten der Worklets angeben zu können, werden mehrere manuelle Analysen, mithilfe des Performance-Tabs der Google Chrome-DevTools, durchgeführt. Dabei konnte festgestellt werden, dass bei den Liniendiagrammen die Ausführungszeiten der Worklets im Schnitt weniger als 0,70ms beträgt. Abhängig von der Anzahl an Datenserien sind die Ausführungszeiten der Worklets bei den Radardiagrammen zwischen 0,60ms und 2,20ms lang. Da es sich bei den Werten der Ausführungszeiten um Millisekunden im niedrigen, einstelligen Bereich bis hin zu Werten im Nullbereich handelt, können die nicht inkludierten Ausführungszeiten der Worklets als geringfügige Messabweichungen bewertet werden.

Liniendiagramm - großer Datensatz

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	65.07 ms	76.02 ms	60.06 ms	669
Chart.js	54.45 ms	59.20 ms	49.70 ms	6
Highcharts	55.72 ms	62.60 ms	51.20 ms	182

Tabelle 5: Messung Liniendiagramm - großer Datensatz, Desktop

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	184.77 ms	278.10 ms	173.80 ms	669
Chart.js	129.16 ms	141.50 ms	123.20 ms	6
Highcharts	127.23 ms	151.60 ms	122.20 ms	72

Tabelle 6: Messung Liniendiagramm - großer Datensatz, Mobile

Die Ergebnisse der Messung „Liniendiagramm - großer Datensatz“ zeigen, dass das Diagramm der Library „HoudiniCharts“ mehr Zeit benötigt, bis es vollständig geladen ist. Die durchschnittliche Ausführungszeit der beiden Referenz-Libraries ist nahezu identisch. Beide benötigen rund 55ms und auf einem Mobilen-Endgerät circa 128ms bis die Liniendiagramme geladen sind. Ebenfalls auffällig ist, dass die Anzahl an DOM-Nodes mit 669 Elementen bei der Library „HoudiniCharts“ deutlich größer ist als bei der Konkurrenz.

Liniendiagramm - kleiner Datensatz

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	19.44 ms	26.80 ms	13.40 ms	49
Chart.js	40.22 ms	47.00 ms	30.00 ms	6
Highcharts	47.86 ms	61.30 ms	37.70 ms	112

Tabelle 7: Messung Liniendiagramm - kleiner Datensatz, Desktop

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	39.40 ms	57.10 ms	35.50 ms	49
Chart.js	80.10 ms	89.60 ms	75.90 ms	6
Highcharts	102.30 ms	118.90 ms	98.10 ms	72

Tabelle 8: Messung Liniendiagramm - kleiner Datensatz, Mobile

Die in Tabelle 7 und 8 dargestellten Resultate veranschaulichen, dass bei einem Liniendiagramm mit wenig Daten die durchschnittliche Ausführungszeit der Library „HoudiniCharts“ mit 19.44ms (Desktop) und 39.40ms (Mobile) signifikant am schnellsten ist. Die Library „Chart.js“ ist mit einer Ausführungszeit von 40.22ms (Desktop) und 80.10ms (Mobile) an zweiter Stelle, gefolgt von „Highcharts“. Außerdem ist zu erwähnen, dass im Vergleich zu dem Testszenario „Liniendiagramm - großer Datensatz“ die Anzahl an DOM-Elementen der Library „HoudiniCharts“ um bis zu 50% geringer ist, als bei der SVG-basierten Library „Highcharts“.

Radardiagramm - großer Datensatz

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	34.42 ms	51.80 ms	20.00 ms	671
Chart.js	86.07 ms	105.20 ms	69.60 ms	5
Highcharts	95.01 ms	118.20 ms	83.30 ms	1085

Tabelle 9: Messung Radardiagramm - großer Datensatz, Desktop

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	59.89 ms	66.40 ms	56.60 ms	671
Chart.js	191.30 ms	207.00 ms	181.60 ms	5
Highcharts	241.28 ms	257.60 ms	232.70 ms	1090

Tabelle 10: Messung Radardiagramm - großer Datensatz, Mobile

Die Ergebnisse der Messung „Radardiagramm - großer Datensatz“ belegen, dass die beiden Libraries „Chart.js“ und „Highcharts“ im Vergleich zu der in dieser Ausarbeitung entwickelten Library „HoudiniCharts“, sowohl bei der Desktop- als auch bei der Mobile-Messung deutlich mehr Zeit benötigen, um die jeweilige Diagramm-Komponente vollständig darzustellen. Die Komponente mit der längsten Ausführungszeit und mit den meisten DOM-Elementen ist erneut von der Library „Highcharts“.

Radardiagramm - kleiner Datensatz

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	15.39 ms	28.07 ms	9.10 ms	41
Chart.js	52.29 ms	66.60 ms	38.40 ms	5
Highcharts	51.10 ms	69.30 ms	34.80 ms	83

Tabelle 11: Messung Radardiagramm - kleiner Datensatz, Desktop

	Ausführungszeit			
Libraries	Durchschnitt	Maximum	Minimum	DOM Größe
HoudiniCharts	26.24 ms	31.60 ms	23.50 ms	41
Chart.js	101.35 ms	127.40 ms	93.00 ms	5
Highcharts	102.53 ms	108.90 ms	97.80 ms	69

Tabelle 12: Messung Radardiagramm - kleines Datenset, Mobile

Bei der Messung „Radardiagramm - kleiner Datensatz“ kann, anhand der in den Tabellen 11 und 12 dargestellten Datensätze, die Aussage getroffen werden, dass die beiden Referenz-Libraries im Rahmen der durchgeführten Messung eine ähnliche Performance aufweisen. Die Library „HoudiniCharts“ ist, wie bereits bei den Ergebnissen der Messung „Radardiagramm - großer Datensatz“ erwähnt, erheblich performanter. Die Anzahl an DOM-Elementen ist erneut bei der Diagramm-Komponente der Library „Highcharts“ am höchsten.

Vergleich Performance

	Durchschnittliche Ausführungszeit			
	Liniendiagramm		Radardiagramm	
Datensatz	Groß	Klein	Groß	Klein
Chart.js	+ 19.50 %	- 51.66 %	- 60.00 %	- 70.57 %
Highcharts	+ 16.76 %	- 59.37 %	- 63.77 %	- 69.88 %

Tabelle 13: Performancevergleich zu „HoudiniCharts“ - Desktop

	Durchschnittliche Ausführungszeit			
	Liniendiagramm		Radardiagramm	
Datensatz	Groß	Klein	Groß	Klein
Chart.js	+ 43.06 %	- 50.82 %	- 68.69 %	- 74.11 %
Highcharts	+ 45.22 %	- 61.49 %	- 75.18 %	- 74.41 %

Tabelle 14: Performancevergleich zu „HoudiniCharts“ - Mobile

In den Tabellen 13 und 14 werden die durchschnittlichen Ausführungszeiten der Library „HoudiniCharts“ im Vergleich zu den beiden Libraries „Chart.js“ und „Highcharts“ prozentual dargestellt. Anhand der Tabellen kann festgestellt werden, dass bei acht von zehn gemessenen Testszenarien die Diagramm-Komponenten von „HoudiniCharts“ zwischen 50% und 75% performanter sind, als die Diagramm-Komponenten der Konkurrenz. Allein bei dem Testszenario „Liniendiagramm - großer Datensatz“ sind die Diagramm-Komponenten der anderen Anbieter im Schnitt 18% und bei der Mobile-Messung circa 44% schneller. Ein Grund hierfür ist, dass bei der entwickelten Library für jeden in der Konfiguration angegebenen Datensatz ein Datenpunkt, in Form eines HTML „<button>“ Elements, erstellt wird. Wie bereits in Kapitel 5.1 beschrieben, enthält das für diese Messung verwendete Datenset, 614 Einträge. Dementsprechend müssen 614 HTML-Elemente dynamisch generiert und in das DOM gerendert werden. Bei der Analyse des SVG-basierten Liniendiagramms der Library „Highcharts“ konnte festgestellt werden, dass ab einer bestimmten Datenmenge auf die Darstellung der einzelnen Datenpunkte verzichtet wird. Aus diesem Grund ist bei der Messung „Liniendiagramm - großer Datensatz“ die DOM-Größe der Diagramm-Komponente mit 182 Elementen deutlich geringer, als bei der Komponente von „HoudiniCharts“ mit 669 Elementen. Die Library „Chart.js“ besitzt diese Problematik nicht, da sie

Canvas-basiert ist und somit nicht alle Elemente einzeln in das DOM gerendert werden müssen.
Zusammenfassend kann, in Bezug auf die Ergebnisse der durchgeführten Performancemessungen, die Aussage getroffen werden, dass die im Rahmen dieser Ausarbeitung hervorgegangene Library „HoudiniCharts“ eine konkurrenzfähige Alternative zu den bereits etablierten Diagramm-Libraries „Chart.js“ und „Highcharts“ darstellt. „HoudiniCharts“ eignet sich besonders für die performante Erstellung von Radardiagrammen jeglicher Art. Sowohl bei den Messungen mit vielen als auch bei den Messungen mit wenigen Daten, waren die Radardiagramm-Komponenten von „HoudiniCharts“ signifikant am schnellsten. Darüber hinaus eignet sich „HoudiniCharts“ optimal für die Visualisierung von Liniendiagrammen mit einer geringen Menge an Daten. Für Liniendiagramme mit einer großen Menge an Daten eignet sich „HoudiniCharts“ hingegen nicht. Hierfür haben sich die Diagramm-Komponente der Library „Chart.js“ bewährt. Unter den getesteten Anbietern performen die SVG-basierten Diagramme der Library „Highcharts“ am schlechtesten. Zusätzlich konnte durch das Verwenden von Paint-Worklets das Ziel erreicht werden, die Menge an Elementen innerhalb des DOMs, im Vergleich zu den SVG-basierten Diagrammen der Library „Highcharts“, zu reduzieren.

5.3 Interaktivität

Im Rahmen dieser Ausarbeitung, konnten die in Unterkapitel 3.3 definierten Features, zur interaktiven Gestaltung der entwickelten Diagramme-Komponenten, umgesetzt werden. Im Folgenden werden die Features „Tooltip“ und „Highlight-Datapoint“ anhand von Screenshots, der aus dieser Ausarbeitung hervorgegangenen Diagramm-Komponenten, visualisiert. Abschließend wird die Interaktivität der Library „HoudiniCharts“ mit den beiden Libraries „Chart.js“ und „Highcharts“ verglichen.

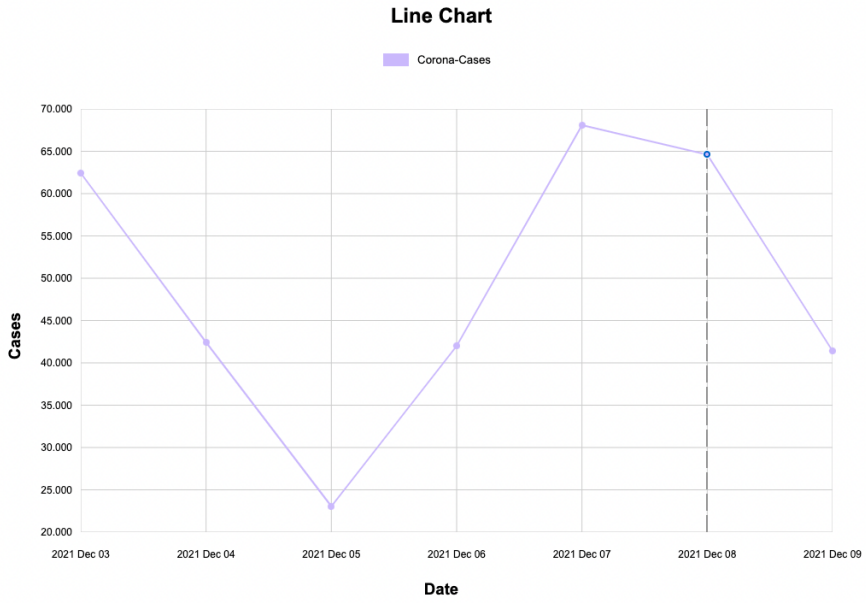


Abbildung 14: Screenshot - Liniendiagramm, Highlight

Der in Abbildung 14 dargestellte Screenshot zeigt ein Liniendiagramm mit dem Feature „Highlight-Datapoint“. Klickt der Nutzer auf einen der Datenpunkte, wird die Position des jeweilige Punktes über eine gestrichelte Gitternetzlinie hervorgehoben.

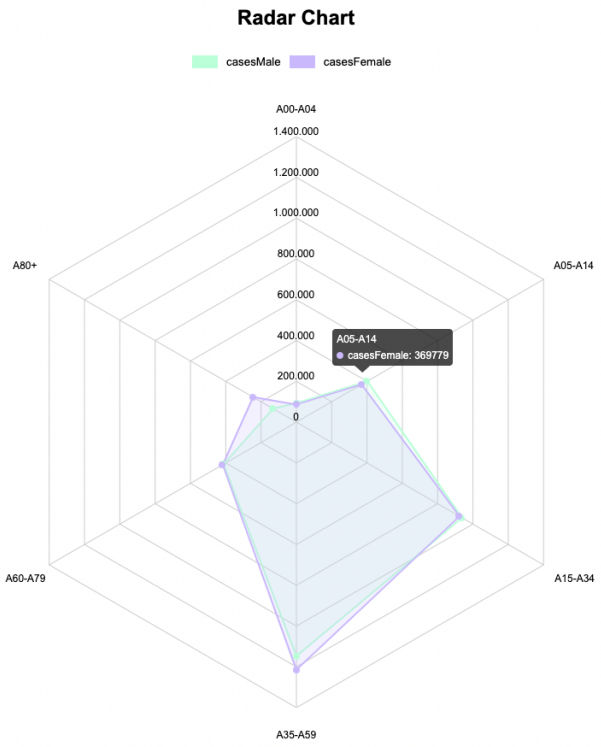


Abbildung 15: Screenshot - Radardiagramm, Tooltip

Das in Abbildung 15 visualisierte Radardiagramm zeigt das Feature „Tooltip“. Hovert der Nutzer mit der Maus über einen der Datenpunkte, wird ihm ein Tooltip mit zusätzlichen Informationen zu dem jeweiligen Datenpunkt angezeigt.
Die entwickelten Features sind keine Alleinstellungsmerkmale der Library „HoudiniCharts“, sondern sind, mit Ausnahme des Features „Highlight-Datapoint“, auch in den beiden Referenz-Libraries standardmäßig implementiert. Das Feature „Highlight-Datapoint“ kann schätzungsweise mit einem Entwicklungsaufwand von circa acht Stunden in den Libraries „Chart.js“ und „Highcharts“ umgesetzt werden.
Ein Vorteil, welcher bei der Erstellung der Library „HoudiniCharts“ festgestellt werden konnte besteht darin, dass im Vergleich zu der Canvas-basierten Library „Chart.js“ Eventlistener direkt dem gewünschten HTML-Element hinzugefügt werden können. Außerdem können durch ein Event ausgelöste Veränderungen, wie zum Beispiel das in Abbildung 14 dargestellte Hervorheben eines Datenpunktes, direkt in den Hintergrund des Diagramms gezeichnet werden. Bei der SVG-basierten Library „Highcharts“ müsste hierfür ein neues Element in das DOM gerendert werden. Ein Nachteil von „HoudiniCharts“ ist, dass bei einer Veränderung des Viewports durch einen Nutzer automatisch alle Paint-Worklets neu gezeichnet werden müssen, wohingegen sich bei den Diagrammen der Library „Highcharts“ der Inhalt ohne sichtbare Veränderungen an den neuen Viewport anpasst.

5.4 Barrierefreiheit

Die Grundlage für die Evaluation der Barrierefreiheit bilden die in der Webanwendung unter der Kategorie „A11y“ aufgelisteten Diagramm-Komponenten. Für die Bewertung der Barrierefreiheit sind zwei Kriterien entscheidend. Zum einen müssen die Diagramme zugänglich für Screenreader sein und zum anderen sollen die Diagrammінhalte über die Steuerung mit der Tastatur erreichbar sein. Die Zugänglichkeit für Screenreader wird mithilfe des macOS „VoiceOver“-Tools überprüft. Für die Bewertung der Barrierefreiheit werden manuelle Tests durchgeführt. Die bereits genannten Bewertungskriterien sind erfüllt, sobald die folgenden Aussagen auf die Diagramm-Komponenten einer Library zutreffen:

1. Alle Diagramm-Informationen können von einem Screenreader erfasst und vorgelesen werden.

2. Jedes interaktive Element in einem Diagramm kann mit der Tastatur angesteuert werden.

Bei den beiden Libraries „HoudiniCharts“ und „Highcharts“ kann über das Konfigurations-Objekt einer Diagramm-Komponente festgelegt werden, ob die Komponente bar-

rierefrei gestaltet werden soll oder nicht. Zusätzlich kann einem Diagramm ein allgemeiner Beschreibungstext hinzugefügt werden. Die Library „Chart.js“ verfügt über kein Feature, um ein Diagramm barrierefrei darzustellen. Grund hierfür ist, dass „Chart.js“ Canvas-basiert ist und somit die Diagramm-Elemente nicht Teil des DOM sind, sondern in ein Canvas-Element gezeichnet werden.
Bei der Analyse der einzelnen Diagramm-Komponenten konnte festgestellt werden, dass die Diagramminhalte sowohl bei der Library „HoudiniCharts“, als auch bei der Library „Highcharts“ optimal mit einem Screenreader erfasst werden können. Außerdem können alle interaktiven Elemente der Diagramm-Komponenten über die Tastatursteuerung erreicht werden. Die Library „Chart.js“ konnte aufgrund des fehlenden Barrierefreiheit-Features nicht analysiert werden.
Abschließend kann zusammengefasst werden, dass die in Unterkapitel 3.4 definierten Anforderungen erfolgreich umgesetzt werden konnten. Sowohl die SVG-basierte Library „Highcharts“, als auch die im Rahmen dieser Ausarbeitung entworfene Library „HoudiniCharts“ eignen sich hervorragend für die Erstellung von barrierefreien Diagrammen. Aufgrund der Tatsache, dass die Struktur der Diagramm-Komponenten der Library „HoudiniCharts“ ausschließlich auf HTML5-Elementen basiert, ist die Semantik der Komponenten, im Vergleich zu den Komponenten der Library „Highcharts“, strukturierter und somit aussagekräftiger. Die Library „Chart.js“ eignet sich nicht für die Erstellung von barrierefreien Diagrammen. Um die Diagramme von „Chart.js“ zugänglich für Screenreader zu gestalten, kann die Struktur eines Diagramms mit zusätzlichen HTML „<table>“ Elementen nachgebaut werden. Dies hat allerdings zur Folge, dass ein Mehraufwand in der Entwicklung des jeweiligen Diagramms entsteht.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit ist es gelungen die JavaScript-Library „HoudiniCharts“ zu entwickeln. Die Library „HoudiniCharts“ ermöglicht es performante, interaktive und barrierefreie Diagramme für das Web zu erstellen. Aktuell unterstützt „HoudiniCharts“ sowohl Linien- als auch Radardiagramme. Die gerenderten Diagramm-Komponenten der Library werden mithilfe von HTML5-Elementen und mittels den in Kapitel 4.1 beschriebenen Schnittstellen von CSS Houdini dargestellt. Zusätzlich wurde im Zuge der Ausarbeitung eine Webanwendung entwickelt, in welcher die Diagramme der Library „HoudiniCharts“ in verschiedenen Kontexten dargestellt werden.
Basierend auf den Ergebnissen der Performancemessungen kann belegt werden, dass die entworfene Library in acht der zehn durchgeführten Testszenarien zwischen 50% und 75% performanter ist, als die beiden Referenz-Libraries „Charts.js“ und „Highcharts“. Neben der herausragenden Performance eignen sich die Komponenten der Library „HoudiniCharts“, aufgrund ihres Aufbaus und den verwendeten Technologien, optimal für die Darstellung von barrierefreien Diagrammen. Darüber hinaus konnten alle für die Library geplanten, interaktiven Features umgesetzt werden. Anhand der aufgeführten Ergebnisse sind die Aussagen fundiert, dass sich CSS Houdini für die Erstellung von interaktiven und performanten Diagrammen zur Datenvisualisierung im Web eignet und dass CSS Houdini damit in Zukunft zur Umsetzung ähnlicher Einsatzzwecke absolut in Betracht gezogen werden kann. Zusätzlich wurde mit der Library „HoudiniCharts“ eine nutzbringende Alternative zu Canvas- und SVG-basierten Diagramm-Libraries entworfen. Neben den signifikanten Vorteilen der entwickelten Library „HoudiniCharts“ bestehen weitere Optimierungsvorschläge und potentielle Erweiterungsmöglichkeiten, welche im folgenden thematisiert werden.
Die in Unterkapitel 5.2.1 aufgeführte Problematik, der vorerst eingeschränkten Browserkompatibilität von „HoudiniCharts“, kann mithilfe des bereits für die Browser Safari und Mozilla Firefox entwickelten „Paint-API“-Polyfills gelöst werden. Zusätzlich sollten Teile des Programmiercodes, in dem das „Typed Object Model API“ oder das „Properties and Values API“ verwendet werden, durch standardmäßige JavaScript-Methoden ersetzt werden. Dadurch kann eine vollständige Kompatibilität mit allen modernen Browsern gewährleistet werden. Allerdings kann infolgedessen die Performance der Library, in den nicht Chromium-basierten Browsern, beeinträchtigt werden. Um dies

zu analysieren sollten weitere Performancemessungen durchgeführt werden, damit Unterschiede in den Ausführungszeiten, aufgrund der jeweiligen Engine des Browsers oder aufgrund des modifizierten Programmiercodes, festgestellt werden können.
Eine potentielle Erweiterung für die Library „HoudiniCharts“ ist die Unterstützung von weiteren Diagrammtypen und Optionen zur Gestaltung der Diagramme. Für die Implementierung von weiteren Diagrammarten kann der in Kapitel 4.4.3 beschriebene generische Aufbau einer Diagramm-Komponente als Leitfaden herangezogen werden. Darüber hinaus könnte ein Feature entwickelt werden welches es ermöglicht, in der Konfiguration einer Diagramm-Komponente, benutzerdefinierte Animationen für das entsprechende Diagramm festzulegen. Dadurch könnten die Diagramme der entwickelten Library visuell attraktiver gestaltet werden. In diesem Kontext lässt sich zudem die Verwendung und die Performance von Animation-Worklets untersuchen. Animation-Worklets sind ein Teil des Sammelbegriffs CSS Houdini und werden zum Zeitpunkt der Erstellung dieser Ausarbeitung nur von Chromium-basierten Browsern unterstützt, bei denen zusätzlich in den Einstellungen die entsprechenden experimentellen Features aktiviert sind. Eine weitere Schnittstelle von CSS Houdini, die im Zusammenhang mit der entwickelten Library überprüft werden kann, ist die „Layout API“. Die „Layout API“ könnte dazu eingesetzt werden eigene Layouts für die Diagramm-Komponenten zu definieren. Dadurch lässt sich möglicherweise die Anzahl der HTML-Elemente, die benötigt werden um ein Diagramm darzustellen, minimieren.

Literaturverzeichnis

[ABR] Atkins-Bittner, Tab ; Remy, François: *CSS Typed OM Level 1 Editor's Draft, 13 October 2021*. Website. https://drafts.css-houdini.org/css-typed-om/, Abruf: 10.11.2021

[amC] amCharts: *JavaScript charting library - amCharts 4*. Website. https://www.amcharts.com/javascript-charts/, Abruf: 30.10.2021

[Ape] ApexCharts: *ApexCharts.js - Open Source JavaScript Charts for your website*. Website. https://apexcharts.com/, Abruf: 30.10.2021

[Bec] Bece, Adrian: *A Practical Overview Of CSS Houdini*. Website. https://www.smashingmagazine.com/2020/03/practical-overview-css-houdini/, Abruf: 08.11.2021

[Bra] Brandt, Mathias: *Bye Bye Flash!* Website. https://de.statista.com/infografik/3798/flash-nutzung-von-webseiten/, Abruf: 16.12.2021

[Cha] Chart.js: *Chart.js, Open source HTML5 Charts for your website*. Website. https://www.chartjs.org/, Abruf: 30.10.2021

[Fen17] Fenton, Steve: *Pro TypeScript - Application-Scale JavaScript Development*. New York : Apress, 2017. – ISBN 978–1–484–23249–1

[Fou] Foundation, Apache S.: *Apache ECharts*. Website. https://echarts.apache.org/en/index.html, Abruf: 30.10.2021

[Gas] Gasperowicz, Ewa: *OffscreenCanvas — Speed up Your Canvas Operations with a Web Worker*. Website. https://developers.google.com/web/updates/2018/08/offscreen-canvas, Abruf: 04.11.2021

[GB] Greif, Sacha ; Benitte, Raphaël: *State of JS 2020: JavaScript Flavors*. Website. https://2020.stateofjs.com/en-US/, Abruf: 10.11.2021

[Gla90] Glassner, Andrew S.: *Graphics Gems*. Stanford : Elsevier Science, 1990. – ISBN 978–0–122–86166–6

[Gooa] Google: *Architecture*. Website. https://github.com/GoogleChrome/lighthouse/blob/master/docs/architecture.md#components--terminology, Abruf: 19.11.2021

[Goob] Google: *Lighthouse*. Website. https://developers.google.com/web/tools/lighthouse, Abruf: 04.12.2021

[Hig] Highcharts: *Highcharts Javascript Charting Library*. Website. https://www.highcharts.com/blog/products/highcharts/, Abruf: 30.10.2021

[Igl19] IGLESIAS, Marcos: *Pro D3.js - Use D3.js to Create Maintainable, Modular, and Testable Charts*. New York : Apress, 2019. – ISBN 978–1–484–25203–1

[Joh] JOHN, Potter: *npmtrends - @amcharts/amcharts4 vs apexcharts vs chart.js vs echarts vs highcharts vs recharts*. Website. https://www.npmtrends.com/apexcharts-vs-chart.js-vs-echarts-vs-highcharts-vs-recharts-vs-@amcharts/amcharts4, Abruf: 30.10.2021

[KJ] KILPATRICK, Ian ; JACKSON, Dean: *CSS Painting API Level 1, Editor's Draft, 15 December 2020*. Website. https://drafts.css-houdini.org/css-paint-api-1/, Abruf: 09.11.2021

[LAS10] LUBBERS, Peter ; ALBERS, Brian ; SALIM, Frank: *Pro HTML5 Programming - Powerful APIs for Richer Internet Application Development*. New York : Apress, 2010. – ISBN 978–1–430–22791–5

[Lib18] LIBBY, Alex: *Beginning SVG - A Practical Introduction to SVG using Real-World Examples*. New York : Apress, 2018. – ISBN 978–1–484–23760–1

[Lü] LÜCKERT, Marlon: *Robert Koch-Institut API - by Marlon Lückert*. Website. https://api.corona-zahlen.org/docs/, Abruf: 21.12.2021

[Mica] MICROSOFT: *HTML5 Canvas and the Canvas Shadow DOM*. Website. https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/hh968259(v=vs.85)?redirectedfrom=MSDN, Abruf: 04.11.2021

[Micb] MICROSOFT: *TypeScript: JavaScript With Syntax For Types.* Website. https://www.typescriptlang.org/, Abruf: 10.11.2021

[Moz] MOZILLA: *Using CSS custom properties (variables)*. Website. https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties, Abruf: 09.11.2021

[Net] NETLIFY: *Netlify: Develop & deploy the best web experiences in record time*. Website. https://www.netlify.com/, Abruf: 12.11.2021

[npm] NPM, Inc.: *About npm*. Website. https://www.npmjs.com/, Abruf: 30.10.2021

[Rec] RECHARTSGROUP: *Recharts*. Website. https://recharts.org/en-US/, Abruf: 30.10.2021

[Sta12] STAPELKAMP, Torsten: *Informationsvisualisierung - Web - Print - Signaletik. Erfolgreiches Informationsdesign: Leitsysteme, Wissensvermittlung und Informationsarchitektur*. Berlin Heidelberg New York : Springer-Verlag, 2012. – ISBN 978–3–642–02076–6

[Sura] SURMA: *Houdini: Demystifying CSS*. Website. https://developers.google.com/web/updates/2016/05/houdini, Abruf: 09.11.2021

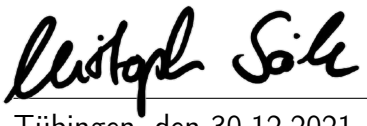
[Surb] Surma: *Is Houdini Ready Yet?* Website. https://ishoudinireadyyet.com/, Abruf: 08.11.2021

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.



Tübingen, den 30.12.2021 Christoph Saile