

Entwicklung eines Verdrahtungseditors in JavaScript auf Basis einer CircuitJS-Simulation

Studienarbeit

des Studiengangs Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Mike Spitzmüller

Abgabe am
14.04.2023

Bearbeitungszeitraum:	09.01.2023 - 14.04.2023
Matrikelnummer, Kurs:	1969093, TEL20AT1
Ausbildungsfirma:	HOBART GmbH
Betreuer der Dualen Hochschule:	Prof. Dr. Rüdiger Heintz

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: „Entwicklung eines Verdrahtungseditors in JavaScript auf Basis einer CircuitJS-Simulation“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Kurzfassung

Die Digitalisierung hat den Lehrprozess in der Hochschulbildung verändert und eröffnet damit neue Möglichkeiten zur Verbesserung des Lernerfolgs von Studierenden. Eine dieser Möglichkeiten besteht darin, interaktive Anwendungen in die Vorlesungen zu integrieren, um sie als mediale Unterstützung in Vorlesungen oder als Übungsplattform zu nutzen.

Diese Studienarbeit dokumentiert die Entwicklung einer Webanwendung zur Verdrahtung von einfachen elektrischen Schaltung in JavaScript. Dieser Verdrahtungseditor soll elektrische Schaltungen auf Basis einer CircuitJS-Anwendung simulieren und darstellen.

Hierfür werden zunächst die nötigen technischen Grundlagen beleuchtet. Dabei richtet sich der Fokus auf die verwendeten Programmiersprachen sowie die Entwicklungsumgebung Visual Studio Code, welche die Basis für den Aufbau der Webanwendung bietet.

Im nächsten Schritt folgt die Analyse des aktuellen Stands der Technik. Hier werden zwei Projekte, welche in vorhergehenden Studienarbeit erarbeitet wurden, erläutert und mögliche Funktionalitäten, die für den Aufbau des Verdrahtungseditor weiterführend sind, analysiert.

Auf Basis dieser bestehenden Projekte wird der Verdrahtungseditor als Frontend-Webanwendung entwickelt. Das Ausgangsprojekt, auf dem dieser aufgebaut wird, ist das Projekt des Logikeditors in der Version 2.0.6. In diesem Projekt erfolgt die Implementierung eines Teils des virtuellen Oszilloskops. Hierbei wird der Fokus auf die bestehende eingebettete CircuitJS-Simulation gelegt, die für den Verdrahtungseditor benötigt wird. Des Weiteren erfolgt die Beschreibung der Umsetzung einige Anpassungen im Projekt sowie der Implementierung neuer Funktionalitäten, wie beispielsweise dem Erstellen neuer Bausteinklassen, der Entwicklung eines Overlays zur Konfiguration der Bausteine sowie einer neuen Menüleiste.

Abschließend wird der Aufbau des Projektes in einem Fazit erläutert und ein Ausblick auf mögliche Weiterentwicklungen des Projektes gegeben.

Abstract

The digitization has changed the teaching process in higher education, opening up new possibilities for improving students' learning success. One of these possibilities is to integrate interactive applications into lectures to use them as media support or as a platform for exercises.

This thesis documents the development of a web application for wiring simple electrical circuits in JavaScript. This wiring editor simulates and displays electrical circuits based on a CircuitJS application.

First, the necessary technical foundations are examined, focusing on the programming languages used and the Visual Studio Code development environment, which provides the basis for building the web application.

The next step is to analyze the current state of the art. Here, two projects developed in previous studies are explained, and possible functionalities that are relevant for the wiring editor's development are analyzed.

Based on these existing projects, the wiring editor is developed as a frontend web application. The starting point for this is the logic editor project version 2.0.6, on which the wiring editor is built. The implementation focuses on the existing embedded CircuitJS simulation, which is required for the wiring editor. Furthermore, the implementation of some modifications in the project and the development of new functionalities, such as creating new component classes, developing an overlay for component configuration, and creating a new menu bar, are described. Finally, the project's structure is explained in a conclusion, and an outlook on possible further developments of the project is given.

Inhaltsverzeichnis

Inhaltsverzeichnis	IV
Abbildungsverzeichnis	V
Listingverzeichnis	VI
1 Einführung	1
1.1 Hintergrund der Arbeit	1
1.2 Aufgabenstellung und Vorgehensweise	1
2 Technische Grundlagen	3
2.1 Visual Studio Code	3
2.2 Hypertext Markup Language	3
2.3 Cascading Style Sheets	4
2.4 JavaScript	4
3 Stand der Technik	6
3.1 Logikeditor	6
3.2 Virtuelles Oszilloskop	10
4 Umsetzung	14
4.1 Implementierung der bestehenden Projekte	14
4.2 Bausteinklassen	16
4.3 Menüleiste und Einbindung der Simulation	18
4.4 Kontextmenü	19
4.5 Overlay zur Konfiguration der Bausteine	20
4.6 Anpassung der CircuitJS-Simulation	24
5 Fazit und Ausblick	26
Literaturverzeichnis	I

Abbildungsverzeichnis

3.1	Übersicht der Dateien des Logikeditor-Projektordners	6
3.2	Benutzeroberfläche der Logikeditor-Webanwendung	7
3.3	Kontextmenü beim Rechtsklicken eines Logikelements	9
3.4	Übersicht der Dateien des virtuellen Oszilloskop-Projektordners	10
3.5	CircuitJS-Simulation der Oszilloskop-Webanwendung	11
3.6	Setzen der Frequenz und Amplitude für externe Sinusspannung	13
4.1	Übersicht der Dateien des entwickelten Verdrahtungseditors	15
4.2	Struktur der Klassen der Funktionsbausteine des Verdrahtungseditors	17
4.3	Schaltflächen der Menüleiste des CircuitJS-Verdrahtungseditors	18
4.4	Eingabe der Frequenz und Amplitude für einen Generator-Baustein	24

Listings

3.1	Konstruktor der Klasse module	8
3.2	iFrame-Element zum Öffnen/Schließen der CircuitJS-Simulation	12
3.3	Funktion didStep(sim) - Anzeige der aktualisierten Daten der Simulation	12
3.4	Speichern der Frequenz und Amplitude in Variablen	13
3.5	Berechnung der externen Sinus-Wechselspannung für Simulation	13
4.1	Einbinden der relevanten JavaScript-Dateien in das Projekt	15
4.2	Image-Attribut beim Erstellen des SVG-Elements	16
4.3	Konstruktorfunktion der Unterklasse oscilloscope	17
4.4	Implementierung der Schaltflächen in die Menüleiste (1)	18
4.5	Implementierung der Schaltflächen in die Menüleiste (2)	18
4.6	Array für das Kontextmenü eines Generators	19
4.7	Funktionsaufruf des d3-context-menu Plugins	20
4.8	Render-Funktion der Klasse Overlay	20
4.9	Funktionsaufruf der Klasse Overlay zum Erstellen der Eingabefenster	20
4.10	Show-Funktion der Klasse Overlay	21
4.11	Erstellen des HTML-Elements für das Auswahlfeld	22
4.12	Setzen der festgelegten Frequenz und Amplitude des Generators	22
4.13	Setzen der ausgewählten Schaltung durch if-Abfrage	22
4.14	Setzen der Eingangsspannung der CircuitJS-Simulation	25
4.15	Laden der Simulation mit neuer Simulationsdatei	25

Abkürzungsverzeichnis

HTML Hypertext Markup Language

CSS Cascading Style Sheets

SVG Scalable Vector Graphics

PNG Portable Network Graphics

1 Einführung

1.1 Hintergrund der Arbeit

Die Digitalisierung hat den Lehrprozess in der Hochschulbildung verändert und eröffnet damit neue Möglichkeiten zur Verbesserung des Lernerfolgs von Studierenden. Eine dieser Möglichkeiten besteht darin, interaktive Anwendungen in die Vorlesungen zu integrieren, um sie als mediale Unterstützung in Vorlesungen oder als Übungsplattform zu nutzen. Des Weiteren sollen zukünftig auch Labore des Studiengangs Elektrotechnik der Dualen Hochschule Baden-Württemberg Mannheim digitalisiert werden, um den Studierenden neue Lernmöglichkeiten zu bieten und ihnen grundlegende elektrische Schaltungen und Messverfahren näher zu bringen. Durch das geplante virtuelle Labor entfällt die Notwendigkeit einer großen Anzahl teurer Hardwarekomponenten. Außerdem wird das Risiko, diese Komponenten bei falscher Benutzung zu beschädigen, minimiert. [1]

Die folgende Studienarbeit beschreibt die Entwicklung eines Verdrahtungseditors für einfache elektrische Schaltungen. Um eine hohe Benutzerfreundlichkeit sowie eine präzise Simulation zu gewährleisten, wird das Projekt als Webanwendung umgesetzt. Als Basis für die Simulation elektronischer Schaltung wird der Schaltungssimulator CircuitJS verwendet, welcher mithilfe von vorherigen Projekten in die Webanwendung integriert werden kann.

1.2 Aufgabenstellung und Vorgehensweise

Ziel der Arbeit ist die Entwicklung einer Webanwendung zur Verdrahtung einfacher elektronischer Messschaltungen und der zugehörigen Simulation durch CircuitJS. Hierfür werden Ergebnisse einiger bereits abgeschlossener Studienarbeiten von Konstantin Fuchs, Wiebke Albers und Justus Epperlein herangezogen, die sich mit der Entwicklung eines Logikanalysators beschäftigen und das Grundgerüst für eine Webanwendung zur Verdrahtung einzelner Module darstellen [2]. Der Editor soll die Möglichkeit bieten, unterschiedliche Module wie einen Funktionsgenerator, verschiedene elektrische Schaltungen sowie ein Oszilloskop erstellen und miteinander verdrahten zu können. Des Weiteren soll der Funktionsgenerator mit einer Frequenz und einer Amplitude kon-

figuriert werden können. Die Messung und Auswertung des Ausgangssignal der Schaltung soll in einem Oszilloskop erfolgen. Hierzu fanden ebenfalls Vorarbeiten in vergangenen Studienarbeiten statt, in welchen ein virtuelles Oszilloskop sowie Schnittstellen zur CircuitJS-Simulation erarbeitet wurden. [3]

Im Zuge dieser Arbeit sollen diese Vorarbeiten in einem Projekt zusammengefasst und angepasst werden. Außerdem sollen die Oberfläche und die Schnittstellen aus dem Grundgerüst des Logikanalysators entsprechend angepasst werden, um einen vollständig funktionalen Verdrahtungseditor zur Simulation einfacher elektronischer Schaltungen zu erhalten.

Um das Ziel zu erreichen, werden zunächst die benötigten technischen Grundlagen für die Entwicklung einer Webanwendung in JavaScript sowie die verwendete Entwicklungsumgebung und Erweiterungen erläutert. Im nächsten Schritt werden die vorherigen Projekte, die für die Entwicklung des Verdrahtungseditors implementiert werden, analysiert. Auf Basis dieser Projekte wird anschließend der Verdrahtungseditor nach den genannten Vorgaben konzipiert und entwickelt. Dabei wird in erster Linie auf die einzelnen Schritte bei der Entwicklung sowie die unterschiedlichen Bereiche des Editors und der CircuitJS-Simulation eingegangen. Abschließend folgt ein Fazit mit einem Ausblick für die zukünftige Weiterentwicklung der Webanwendung sowie dem darauffolgenden Einsatz im virtuellen Grundlagenlabor.

2 Technische Grundlagen

2.1 Visual Studio Code

„Visual Studio Code ist das neueste Entwicklerwerkzeug von Microsoft“ [4]. Der Quellcode-Editor ist leistungsstark, aber auch leichtgewichtig. Das bedeutet er ist sofort einsatzbereit. Im Gegensatz zu Visual Studio 2019 wird hier mit sogenannten Arbeitsumgebungen gearbeitet, auch Workspaces genannt. In diesen Workspaces wird neben den Änderungen in den Dateien auch der Bearbeitungszustand, die Reihenfolge der geöffneten Dateien und deren Zeilenposition gespeichert, sodass diese Einstellungen wieder aufgerufen werden können. [5]

Neben der neuen Arbeitsweise mit Workspaces bringt das Programm auch Funktionalitäten, wie beispielsweise die Hervorhebung von Syntax, eine Textsuche in Dateien und Ordern oder einen Debugger für verschiedene Applikationen mit sich. Auch eine Unterstützung für JavaScript, TypeScript und Node.js ist in Visual Studio Code vorhanden. Neben diesen integrierten Funktionen werden innerhalb des Programms einige Bibliotheken mit Erweiterungen für andere Programmiersprachen und Laufzeiten geboten. Die Entwicklungsumgebung ist betriebssystemübergreifend und somit auch mit MacOS und Linux kompatibel. [5]

2.2 Hypertext Markup Language

Hypertext Markup Language (HTML) ist eine Auszeichnungssprache zur Strukturierung und Formatierung von Inhalten und somit eine der grundlegenden Technologien für die Erstellung von Webseiten. Die Sprache wurde erstmals im Jahr 1993 von Tim Berners-Lee entwickelt und wird seither kontinuierlich in verschiedenen Version weiterentwickelt. [6, S. 2]

Durch HTML können Entwickler Formatierungen für Texte, Farben, Bilder und Links einfügen, um eine Website benutzerfreundlicher und interaktiver zu gestalten. „Die Beschreibung der einzelnen Komponenten einer Website wie Überschriften, Absätze, Tabellen, Bilder und Links erfolgt in HTML mit speziellen Steueranweisungen“, auch als Tags bezeichnet [6, S. 2]. Die einzelnen Auszeichnungen eines HTML-Dokuments befinden sich zwischen spitzen Klammern. „Jeder Tag hat einen Start-Tag und einen Ende-Tag. Der EndeTag entspricht dem Anfangstag mit einem

vorgestellten / bzw. in der Kurzform als Ende des Starttags `</>` [7, S. 24–25]. Hierbei ist zu beachten, dass die jeweiligen Auszeichnungen in richtiger Reihenfolge geöffnet und geschlossen werden und innerhalb eines `<html>`-Tags stehen. Zwischen Groß- und Kleinschreibung wird bei HTML nicht unterschieden. Dadurch entsteht eine möglichst einfache Auszeichnungssprache, die der Entwickler schnell erlernen und verstehen kann.

Die neueste Version der HTML-Technologie ist HTML-5.2 und bringt neue Elemente zur verbesserten Formatierung von Inhalten mit sich, wie beispielsweise die Anweisungen `<section>` oder `<header>`. Durch neuere HTML-Versionen kann es dadurch bei älteren Browsern zu einer inkorrekten Darstellung der Inhalte kommen. Im Gegensatz zu Vorgängerversionen wurde des Weiteren auch die Dynamik der Version HTML-5.2 verbessert. [6, S. 3]

2.3 Cascading Style Sheets

Cascading Style Sheets (CSS) ist eine Programmiersprache zur Gestaltung von Webseiten und wurde im Jahr 1996 als Ergänzung zu HTML eingeführt. Ein wichtiger Grund für die Einführung von CSS war die dynamische Anpassung von Webseiten an unterschiedliche Displaygrößen der Endgeräte [6, S. 4]. Des Weiteren ermöglicht CSS die Änderung des Aussehens von HTML-Elementen beispielsweise in ihrer Schriftart, Farbe, Größe und Abständen zu anderen Elementen. Entwickler können dadurch auch Animationen, Übergänge oder andere visuelle Effekte in ihre Webseite integrieren.

Neben diesen Ergänzungen bietet CSS viele weitere Vorteile für die Handhabung bei der Entwicklung von Webseiten. Zum einen ermöglicht es eine bessere Trennung von Inhalt und Design. Das bedeutet, dass Entwickler den Inhalt einer Webseite und das Design separat bearbeiten können, was die Wartung und Aktualisierung von Webseiten erleichtert. Ein weiterer Vorteil von CSS ist, dass es die Ladezeiten von Webseiten verkürzt. CSS-Dateien werden in der Regel separat von HTML-Dateien geladen und können in einer Datei zusammengefasst werden. Dadurch können Webseiten schneller geladen werden, da die Größe der HTML-Dateien reduziert wird. [8]

2.4 JavaScript

JavaScript ist eine Programmiersprache, welche im Jahr 1995 von Netscape entwickelt wurde. Der ursprüngliche Name „LiveScript“ wurde im Jahr 1996 durch JavaScript ersetzt, um die damalige Popularität von Java zu nutzen. JavaScript findet hauptsächlich Einsatz in Anwendungen im Webbrowser, da hier die „Hypertext Markup Language“ (HTML) durch die Logik von JavaScript erweitert werden kann und somit interaktive Funktionen der Website ermöglicht werden. In erster Linie wird die Skriptsprache verwendet, um Formulareingaben eines Benutzers überprüfen zu

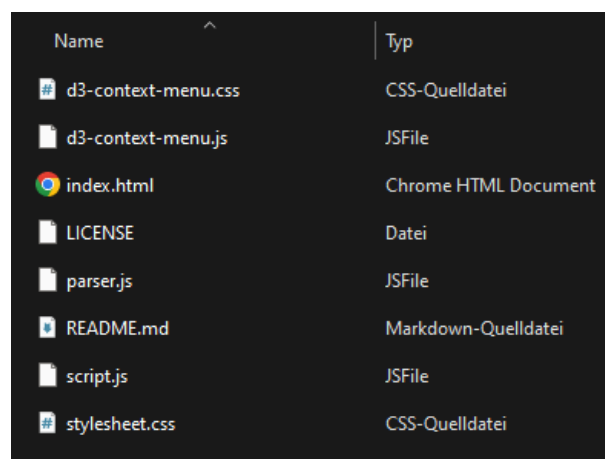
können. Durch den häufigen Einsatz wird die Programmiersprache von allen gängigen Browsern unterstützt. Neben Formulareingaben findet JavaScript auch Anwendung bei komplexeren Kontrollabfragen, welche mit einer Datenbank verknüpft sein können, sowie in der Spieleentwicklung für Browser und Apps für mobile Geräte. [7, S. 79–80] JavaScript ist eine interpretierte Sprache, was bedeutet, dass der Code direkt im Browser, anstatt auf dem Server ausgeführt wird. Dies macht JavaScript schneller als serverseitige Skriptsprachen wie PHP oder Python, da der Browser nicht auf eine Antwort des Servers warten muss, sondern der Code direkt im Browser des Benutzers ausgeführt wird. [7, S. 8]

3 Stand der Technik

Im folgenden Kapitel werden zwei Projekte aus vorhergehenden Arbeiten betrachtet und analysiert. Hierbei wird der Fokus auf den aktuellen Stand der entwickelten Anwendungen gelegt. Des Weiteren wird auf einige wichtige Funktionen eingegangen, die für das Projekt relevant sind, da diese Vorarbeiten im späteren Verlauf ineinander integriert werden sollen, um somit eine vollständig funktionale Anwendung des Verdrahtungseditors zu ermöglichen.

3.1 Logikeditor

Das erste betrachtete Projekt ist ein Logikeditor zur Darstellung von digitaler Logik, welcher in vorhergehenden Studienarbeiten als Frontend Webanwendung umgesetzt wurde [2, S. 1]. Das Projekt setzt sich aus mehreren Hypertext Markup Language (HTML)-, Cascading Style Sheets (CSS)- und JavaScript-Dateien zusammen, wie in der Abbildung 3.1 zu erkennen ist. Des Weiteren befinden sich im Projektordner zwei Textdokumente `LICENCE` und `README`, welche zur groben Beschreibung des Projektes und zur Angabe der Lizenz- und Nutzungsbestimmungen der Anwendung dienen.



Name	Typ
d3-context-menu.css	CSS-Quelldatei
d3-context-menu.js	JSFile
index.html	Chrome HTML Document
LICENSE	Datei
parser.js	JSFile
README.md	Markdown-Quelldatei
script.js	JSFile
stylesheet.css	CSS-Quelldatei

Abbildung 3.1: Übersicht der Dateien des Logikeditor-Projektordners

Für die Darstellung der Anwendung in einem Webbrowser dient die `index.html`-Datei. In dieser findet die Darstellung der Applikation statt, da hier zum einen die Komponenten beispielsweise für Texte und Grafiken beschrieben werden. Zum Anderen wird das Projekt in dieser Datei zusammengefügt und die einzelnen Dateien inkludiert. In diesem Projekt ist eine `stylesheet.css`-Datei vorhanden, die die einzelnen HTML-Komponenten formatiert. Für die Verknüpfung der HTML-Elemente mit der Logik des Logikeditor dienen zwei JavaScript-Dateien. Die `script.js`-Datei beinhaltet die meiste Logik des Projektes. Hier findet die Deklaration unterschiedlicher Klassen und Funktionen, die für den Logikeditor relevant sind, statt. Auf einige Teile der `script.js`-Datei wird im späteren Verlauf der Projektbeschreibung genauer eingegangen. In der `parser.js`-Datei sind verschiedene Funktionen zur Auswertung eines Eingabefenster für die entsprechende Logik deklariert. Da dies für den zukünftigen Verdrahtungseditor nicht relevant ist, wird die Datei nicht genauer erläutert.

Neben diesen Dateien ist das Plugin "d3-context-menu" in diesem Projekt integriert und umfasst eine JavaScript- und eine Cascading Style Sheets (CSS)-Datei. Das Plug-In ermöglicht dem Entwickler das Erstellen verschiedener Kontextmenüs und wird für den Aufbau des Editors benötigt [9].

Die Oberfläche des Logikeditors ist in zwei Bereiche aufgeteilt, der Menüleiste und dem Simulationsfenster, und wird in Abbildung 3.2 mit einem Beispiellogikgatter sowie den geöffneten Dropdownmenüs dargestellt.

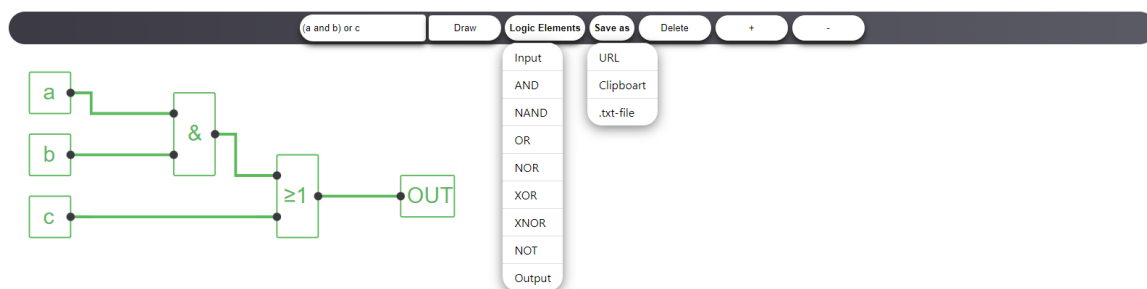


Abbildung 3.2: Benutzeroberfläche der Logikeditor-Webanwendung

Innerhalb der Menüleiste, welche sich am oberen Rand der Anwendung befindet, gibt es verschiedene Schaltflächen und ein Eingabefenster. Hier können beispielsweise einzelne Logikelemente ausgewählt, in das Simulationsfenster rein- oder rausgezoomt oder auch die erstellte Schaltung abgespeichert werden. Das Simulationsfenster, welches sich unterhalb der Menüleiste befindet, ist ein weißer Kasten, in dem die Logikgatter aufgebaut werden können. Dies ist der Arbeitsbereich des Tools. Hier werden die einzelnen Logikblöcke als Scalable Vector Graphics (SVG) gezeichnet. Bei dieser Vorgehensweise zur Beschreibung zweidimensionaler Objekte bestehen diese nicht aus einzelnen Pixeln, sondern werden aus Vektoren zusammengesetzt. Dies bringt den Vorteil mit

sich, dass die Objekte, unabhängig vom Zoom des Simulationsfensters, immer scharf abgebildet werden [10, S. 856]. Die Verbindungen, die zwischen den einzelnen Logikelementen gezeichnet werden, werden ebenfalls als SVG dargestellt. Die Logikelemente können außerdem per Drag-and-Drop Funktion innerhalb des Arbeitsbereichs frei bewegt werden.

Durch diese wesentlichen Elemente des Logikeditors können verschiedene Logikschaltungen sehr einfach und übersichtlich aufgebaut werden. Das Grundprinzip der SVG-Zeichnung von Elementen wird bei der Entwicklung eines Verdrahtungseditors für CircuitJS-Schaltungen ebenfalls benötigt. Aus diesem Grund werden im Folgenden einige wichtige Funktion anhand des jeweiligen Quellcodes beschrieben und erläutert, inwiefern diese Funktionalitäten für den zukünftigen Verdrahtungseditor wiederverwendet werden können.

Die Oberfläche des Logikeditors ist in der `index.html`-Datei beschrieben und durch `<div>`-Elemente in die zwei Bereiche eingeteilt. Hier werden die einzelnen Schaltflächen beschrieben. Mit dem Eventhandler `onclick` wird eine Funktion der `script.js`-Datei aufgerufen, um somit die Schaltfläche mit der entsprechenden Logik zu versehen. Ein Beispiel hierfür sind die einzelnen Schaltflächen des Dropdownmenüs `Logic Elements`. Hier können einzelne Logikelemente ausgezählt und gezeichnet werden. Dafür wird die Funktion `place()` aufgerufen und der Funktion die Klasse des entsprechenden Elements mit einer bestimmten Position übergeben.

Die einzelnen Logikelemente sind als Klassen definiert und mit entsprechenden Parametern beschrieben. Diese sind Unterklassen der übergeordneten Klasse `module`. Die `module`-Klasse ist die Basis der einzelnen Logikelemente. Sie enthält einen Konstruktor (vgl. Listing 3.1), welcher dem Objekt Parameter wie die Dimensionen des Blockes oder der Anzahl an Ein- und Ausgängen zuweist. Des Weiteren enthält diese Klasse die Funktionen, um das SVG-Element zu zeichnen, zu verschieben und weiteres. Dadurch entsteht ein Grundgerüst für die Entwicklung einer interaktiven Anwendung mit SVG-Objekten, welches für die Entwicklung des Verdrahtungseditor übernommen werden kann. Hier müssen dann im späteren Verlauf neue Unterklassen für die einzelnen Blöcke definiert und den Klassen weitere Parameter zugewiesen werden. Einige Parameter und Funktionen der `module`-Klasse können jedoch vernachlässigt werden, da diese spezifisch für die Anwendung mit Logikbausteinen benötigt werden.

```
1      constructor(x, y) {
2          this.classname = this.constructor.name;
3          this.x = nextGridPoint(x);
4          this.y = nextGridPoint(y);
5          this.id = modulID;
6          modulID++;
7          this.width = 30;
8          this.height = 30;
9          this.maxInputCount = 2;
10         this.maxOutputCount = 1;
11         this.outputOffset = [];
12         this.inputOffset = [];
13         this.connectionPointRadius = 3;
```



```
14     this.sizeStrokeWidth = 3;
15     this.formRadius = 1;
16     this.group = null;
17     this.input = new Array();
18     this.output = new Array();
19     this.text = '';
20     this.negation = false;
21     this.value = false;
22     this.selected = false;
23     new ModuleManager().addModule(this);
24 }
25
```

Listing 3.1: Konstruktor der Klasse module

Neben den SVG-Objekten für die Darstellung der Bausteine kann auch die Funktionalität der Verbindungen der Bausteine wiederverwendet werden. Diese ist ebenfalls in einer Klasse `connection` definiert und enthält verschiedene Parameter. In der letzten Version des Logikeditors wurden die Verbindungen angepasst, sodass diese nicht mehr mit einem Bogen gezeichnet werden, sondern geradlinig mit rechten Winkeln. Da der Verdrahtungseditor für elektrische Schaltungen möglichst realitätsnah aufgebaut werden soll, ist eine Anforderung, dass die Verbindungen als Bogen gezeichnet werden. Aus diesem Grund wird als Grundlage für den Aufbau des Verdrahtungseditor nicht die neueste Version des Logikeditors verwendet, sondern die Version 2.0.6.

Mithilfe des „d3-context-menu“ Plugins wurde ein Kontextmenü für die einzelnen Logikelemente erstellt. Beim Rechtsklicken auf den gewählten Baustein öffnet sich ein Menü, in dem unterschiedliche Funktionalitäten implementiert sind (vgl. Abbildung 3.3). Hier kann beispielsweise der Baustein gelöscht oder durch einen anderen ersetzt werden. Dieses Kontextmenü kann ebenfalls in der späteren Anwendung des Verdrahtungseditors verwendet werden, um zum Beispiel die Schaltungen entsprechend zu konfigurieren.

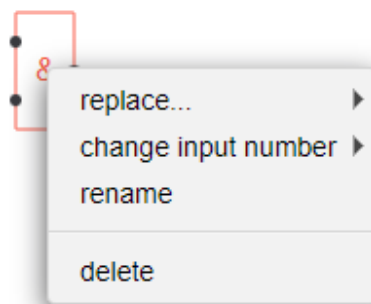


Abbildung 3.3: Kontextmenü beim Rechtsklicken eines Logikelements

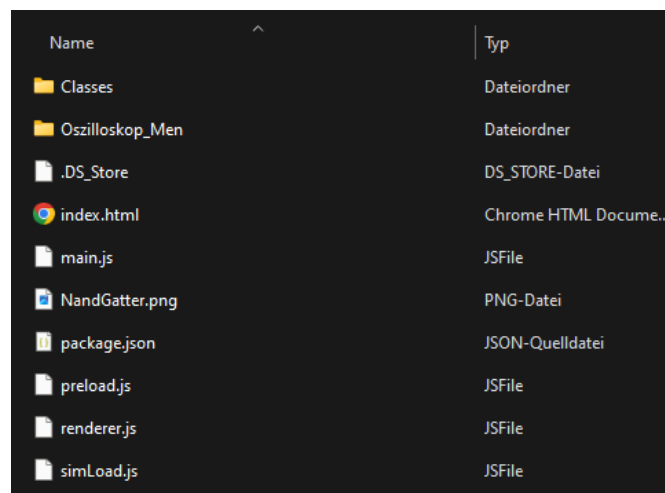
Insgesamt lässt sich sagen, dass die Anwendung eine gute Grundlage bietet, um einen Verdrahtungs-

tungseditor für elektrische Schaltungen aufzubauen. Das Grundgerüst der SVG-Elemente sowie einige Funktionalitäten können übernommen werden. Die Anwendung ist objektorientiert programmiert und bietet somit gute Möglichkeiten zur Weiterentwicklung und zum Implementieren neuer Funktionen.

3.2 Virtuelles Oszilloskop

Um die elektrischen Schaltungen für einen Verdrahtungseditor simulieren zu können, soll eine CircuitJS-Simulation integriert werden. Auch hierfür fanden bereits Vorarbeiten bei der Entwicklung eines virtuellen Oszilloskops statt.

Der Projektordner setzt sich aus einer Vielzahl unterschiedlicher Dateien zusammen. Auch bei dieser Anwendung handelt es sich um eine Frontend Webanwendung, welche auf HTML-, CSS- und JavaScript-Dateien basiert. Der Unterschied zum Projektordner des Logikeditors ist, dass hier mehrere Unterordner vorhanden sind, in welche das Projekt unterteilt ist. Hier gibt es beispielsweise einen Ordner **war**, in dem die Simulationen der Webanwendung CircuitJS gespeichert sind. Die Dateien für das entwickelte Oszilloskop liegen im Ordner **app** ab (vgl. Abbildung 3.4). Hier gibt es ebenfalls eine **index.html**-Datei, welche die Oberfläche der Webanwendung beschreibt und die anderen CSS- und JavaScript-Dateien in diesem Projekt zusammenfügt. Die Logik der Simulation sowie die Deklaration der Klassen und das virtuelle Oszilloskop sind in mehrere JavaScript-Dateien unterteilt, um die Weiterentwicklung der Anwendung übersichtlicher zu gestalten.



Name	Typ
Classes	Dateiordner
Oszilloskop_Men	Dateiordner
.DS_Store	DS_STORE-Datei
index.html	Chrome HTML Docume...
main.js	JSFile
NandGatter.png	PNG-Datei
package.json	JSON-Quelldatei
preload.js	JSFile
renderer.js	JSFile
simLoad.js	JSFile

Abbildung 3.4: Übersicht der Dateien des virtuellen Oszilloskop-Projektordners

Das virtuelle Oszilloskop bietet in der vorliegenden Version sowohl die Möglichkeit zur Darstel-

lung eines einfachen Oszilloskops mit mehreren Eingängen sowie die integrierte Simulation der CircuitJS-Schaltungen. Die Oberfläche der Webanwendung ist in drei Abschnitte unterteilt. Am oberen Rand der Oberfläche befindet sich die Anzeige des virtuellen Oszilloskops, welches mit einem Canvas-Element umgesetzt ist. Das Canvas-Element ist eine programmierbare Fläche, in der beispielsweise Grafiken oder Animationen mithilfe von JavaScript-Code erstellt werden können [11]. Der Unterschied zwischen SVG und dem Canvas-Element ist, dass das Canvas-Element pixelbasiert und nicht vektorbasiert gezeichnet wird und das Bitmap-Bildformat verwendet, wodurch das Ergebnis abhängig von der jeweiligen Auflösung und dem Zoom ist. Im Gegensatz dazu ist das Canvas-Element deutlich flexibler und es gibt mehr Möglichkeiten, die Interaktivität für den Benutzer zu ermöglichen [12]. Das virtuelle Oszilloskop wird aktuell in einer weiterführenden Studienarbeit fertiggestellt und in der Vorbetrachtung nicht weiter berücksichtigt.

Der Fokus der Vorbetrachtung liegt auf der CircuitJS-Simulation, welche sich am unteren Ende der Anwendungsoberfläche befindet (vgl. Abbildung 3.5)

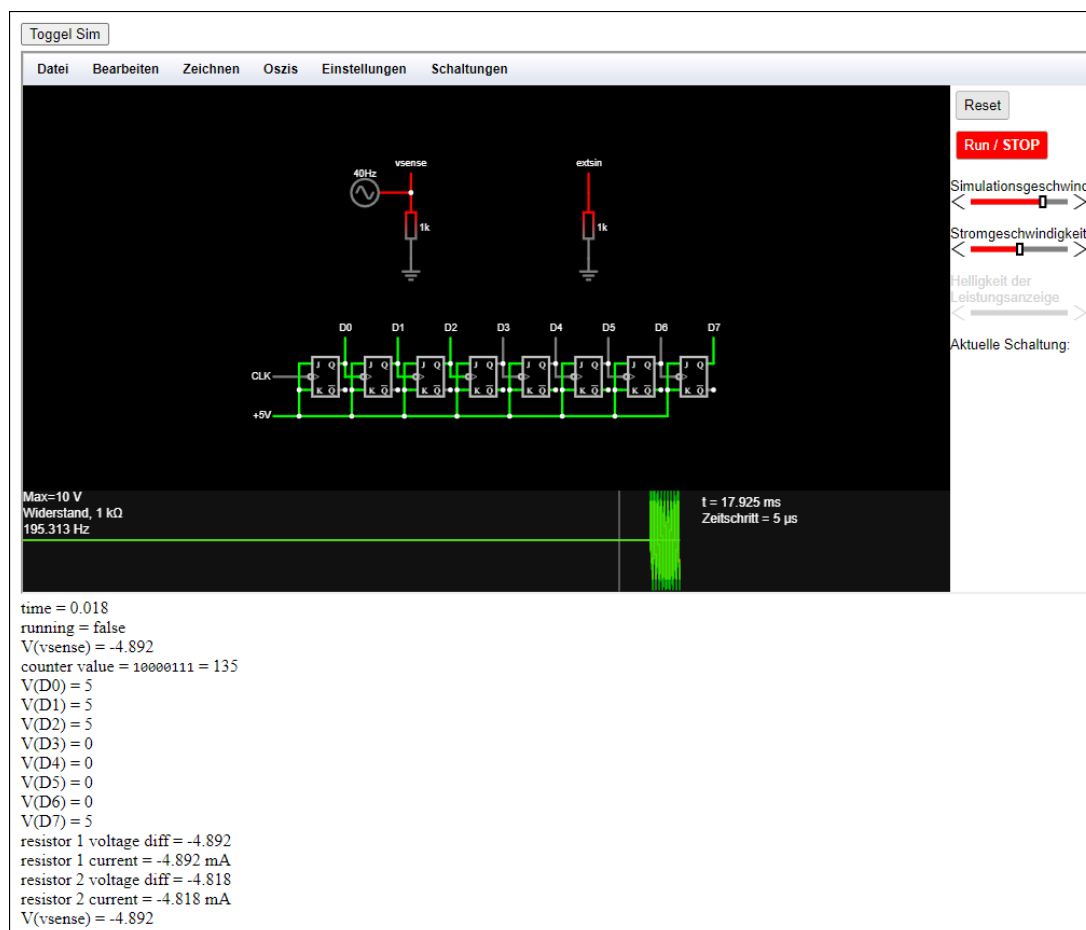


Abbildung 3.5: CircuitJS-Simulation der Oszilloskop-Webanwendung

Innerhalb dieses Simulationsfensters befindet sich die Schaltfläche `Toggle Sim`. Mit dem Event-

handler `onclick`, welcher beim Betätigen der Schaltfläche aktiviert wird, wird die Funktion `toggleIFrame()` aufgerufen. Dadurch wird das `iFrame`-Element und somit auch das `Canvas`-Element, in dem sich die `CircuitJS`-Simulation befindet, wird geschlossen oder geöffnet (vgl. Listing 3.2). Hierbei wird das `iFrame`-Element verwendet, da dies „die Einbettung eines externen Dokuments, wie beispielsweise PDF-Dateien oder anderen Webseiten, in das aktuelle Dokument ermöglicht.“ [13]

```
1 <div style="border: 2px solid black; padding: 10px; margin-top: 8px;">
2 <button id="iFrameToggle" onclick="toggleIFrame()" style="margin-bottom: 5px
">Toggle Sim</button>
3 <iframe id="circuitFrame" width="100%" height="500" position="relative" src="
../war/circuitjs.html?startCircuit=jsinterface.txt" style="display:none;"></
iframe>
4 <script src="simLoad.js"></script>
5 <script>
6     function toggleIFrame() {
7         let iframeDisplayStatus = document.getElementById("circuitFrame").style.
display;
8         if(iframeDisplayStatus == "") {
9             document.getElementById("circuitFrame").style.display = "none";
10        }
11        else {
12            document.getElementById("circuitFrame").style.display = "";
13        }
14    }
15 </script>
```

Listing 3.2: `iFrame`-Element zum Öffnen/Schließen der `CircuitJS`-Simulation

Unterhalb des `iFrame`-Elements mit der Simulation befinden sich weitere `HTML`-Texte. Diese stellen die Schnittstelle zur `CircuitJS`-internen Simulation nach außen dar, da diese beim Start der Simulation in zyklischen Abständen aktualisiert werden. Dies erfolgt durch die JavaScript-Datei `simLoad.js`. Hier wird die Funktion `didStep(sim)` (vgl. Listing 3.3) zyklisch aufgerufen und die Daten der Simulation mit der Funktion `didUpdate(sim)` aktualisiert und angezeigt.

```
1 function didStep(sim) {
2     var t = sim.getTime();
3     var q1 = ampl*Math.sin(freq*Math.PI*2*t);
4     var q2 = ampl*Math.cos(freq*Math.PI*2*t);
5     var q3 = 4;
6     var q4 = 10;
7     document.dispatchEvent(new CustomEvent('DidSimStep',
8     {
9         detail:
10         {
11             time: t,
12             sig1: q1,
13             sig2: q2,
14             sig3: q3,
```

```

15         sig4: q4
16     }
17 }));

```

Listing 3.3: Funktion didStep(sim) - Anzeige der aktualisierten Daten der Simulation

Des Weiteren kann in der Webanwendung eine externe Sinus-Wechselspannung definiert und der CircuitJS-Simulation übergeben werden. Hierfür befinden sich zwei auf der Oberfläche zwei Schaltflächen für die Eingabe der Frequenz und der Amplitude (vgl. Abbildung 3.6).

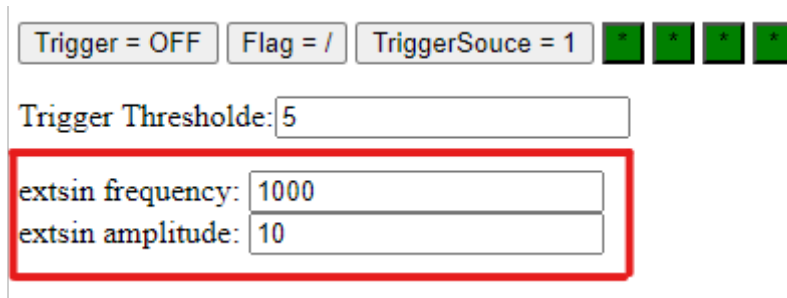


Abbildung 3.6: Setzen der Frequenz und Amplitude für externe Sinusspannung

Diese Schaltflächen werden mit der Methode `getElementById` abgefragt und in entsprechenden Variablen abgespeichert (vgl. Listing 3.4).

```

1     freq = parseFloat(document.getElementById("freq").value);
2     ampl = parseFloat(document.getElementById("ampl").value);

```

Listing 3.4: Speichern der Frequenz und Amplitude in Variablen

In zyklischen Abständen wird mit diesen Variablen anschließend in der Funktion `didStep(sim)` eine Sinus-Wechselspannung berechnet und der Simulation durch den Funktionsaufruf in Listing 3.5 übergeben.

```

1     // set voltage of external voltage "extsin"
2     sim.setExtVoltage("extsin", ampl*Math.sin(freq*Math.PI*2*t));

```

Listing 3.5: Berechnung der externen Sinus-Wechselspannung für Simulation

Die Funktionen der JavaScript-Datei `simLoad.js` ermöglichen die Schnittstelle der CircuitJS-Simulation und können für die Entwicklung des Verdrahtungseditors wiederverwendet und weiterentwickelt werden, da die CircuitJS-Simulation die Grundlage des Verdrahtungseditors darstellen soll. Hierfür kann ein Teil der Funktionalitäten der Webanwendung des virtuellen Oszilloskops in das zukünftige Projekt implementiert und angepasst werden.

4 Umsetzung

Das folgende Kapitel beschreibt die Umsetzung des Verdrahtungseditors für einfache elektrische Schaltungen, welcher auf einer CircuitJS-Simulation basieren soll. Hierfür werden zunächst die in Kapitel 3 beschriebenen Projekte herangezogen und in einem gemeinsamen Projekt zusammengefügt. Anschließend werden einige Anpassungen an den bestehenden Projekten vorgenommen. Des Weiteren werden neue Klassen und Funktionen erstellt und damit ein vollständig funktionierender Verdrahtungseditor entwickelt.

4.1 Implementierung der bestehenden Projekte

Um den Verdrahtungseditor aufzubauen, werden zunächst die beiden bestehenden Projekte in einem gemeinsamen Projekt zusammengefügt. Das Ausgangsprojekt, auf dem der Editor aufgebaut wird, ist die Version 2.0.6 des Logikeditors (vgl. Abschnitt 3.1). In dieses Projekt wird ein Teil des bestehenden virtuellen Oszilloskops, die CircuitJS-Simulation, integriert.

Für die Implementierung der CircuitJS-Simulation in das Logikeditor-Projekt, müssen zunächst alle relevanten Dateien in einer gemeinsamen Ordnerstruktur eingepflegt werden. Hierfür wird der Projektordner des Logikeditors als Basis gewählt und die zugehörigen Dateien der Simulation in diesen eingefügt. Abbildung 4.1 zeigt den zusammengefügten Projektordner für den Verdrahtungseditor. Hierbei ist zu erkennen, dass für die Simulation lediglich die Datei `simLoad.js` sowie der zugehörige Ordner `war` integriert werden. In dem Ordner befinden sich alle relevanten Dateien für die Simulation der CircuitJS-Anwendung. Die `simLoad.js`-Datei stellt hierbei die Schnittstelle zwischen der CircuitJS-Simulation und der Webanwendung dar. Hierauf wird in Kapitel 4.6 genauer eingegangen.

Name	Typ
.vscode	Dateiordner
Circuits	Dateiordner
Classes	Dateiordner
components	Dateiordner
war	Dateiordner
d3-context-menu.css	CSS-Quelldatei
d3-context-menu.js	JSFile
index.html	Chrome HTML Do...
LICENSE	Datei
parser.js	JSFile
README.md	Markdown-Quelld...
script.js	JSFile
simLoad.js	JSFile
stylesheet.css	CSS-Quelldatei
z_notizen.txt	Textdokument

Abbildung 4.1: Übersicht der Dateien des entwickelten Verdrahtungseditors

Damit die Simulation der CircuitJS-Anwendung in das Projekt integriert ist, muss außerdem die Datei `index.html` angepasst werden. Hierfür wurden zunächst alle für die Webanwendungen relevanten JavaScript- und CSS-Dateien eingebunden, wie in Listing 4.1 zu erkennen ist. Hierbei ist darauf zu achten, dass die Einbindung der Skripte im `<head>`-Element erfolgt. Neben den Dateien des Projektordners wurden außerdem die Bibliotheken „d3“ und „jQuery“ eingebunden (vgl. Z. 3-4, Listing 4.1).

Die JavaScript-Dateien für das virtuelle Oszilloskop wurden hier nicht eingebunden. Dies könnte zukünftig erfolgen, um das virtuelle Oszilloskop ebenfalls in die Anwendung einzubinden und zu verwenden (vgl. Kapitel 5).

```

1  <link rel="stylesheet" href="stylesheet.css">
2  <link rel="stylesheet" href="d3-context-menu.css">
3  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.
  js"></script>
4  <script src="https://d3js.org/d3.v5.min.js"></script>
5  <script src="components/overlay/overlay.js"></script>
6  <script src="d3-context-menu.js"></script>
7  <script src="script.js"></script>
8  <script src="./Classes/connection.js"></script>
9  <script src="./Classes/managers.js"></script>
10 <script src="./Classes/modules.js"></script>
11 <script src="simLoad.js"></script>

```

Listing 4.1: Einbinden der relevanten JavaScript-Dateien in das Projekt

4.2 Bausteinklassen

Derzeit befindet sich die komplette Logik des Logikeditors in der JavaScript-Datei `script.js`, darunter auch die verschiedenen Klassen des Projektes. Im ersten Schritt wurden diese Klassen aus der Datei ausgelagert und neue JavaScript-Dateien für die Verbindungen (`connection.js`), die Manager (`managers.js`) und für die Funktionsbausteine (`modules.js`) erstellt.

Die Erzeugung der Verbindungen, welche in der Version 2.0.6 mit einer Kurve zwischen den Modulen erzeugt wird, wird aus dem Logikeditor übernommen. Des Weiteren werden die Manager, welche eine Liste der vorhandenen Verbindungen und Module erstellen, mit in den Verdrahtungseditor integriert, um die einzelnen Module und Verbindungen bearbeiten und löschen zu können. Dafür beinhaltet beispielsweise der `ModuleManager` Funktionen, um die Module hinzuzufügen oder sie zu löschen.

Um die einzelnen Funktionsbausteine für den Verdrahtungseditor umzusetzen, kann als Basis für diese die Klasse `Module` herangezogen werden. Diese Klasse enthält einen Konstruktor, welche einige Klassenvariablen initialisiert, einschließlich der x- und y-Position, der Größe des Moduls, der maximalen Anzahl von Eingängen und Ausgängen sowie der Offsets für die Verbindungsstellen. Des Weiteren beinhaltet die Klasse Funktionen zur Beschreibung und Bearbeitung der Module. Die Funktion `build` erstellt ein SVG-Element, das ein Modul des Editors darstellt, einschließlich der Rechteckform des Moduls, der Größe und den Verbindungspunkten für Ein- und Ausgänge. Hier wurde der Text, welcher zuvor in das SVG-Element eingefügt wurde, durch ein `Image`-Attribut ersetzt. Dadurch kann für das erstellte Modul ein Bild gesetzt werden, welches innerhalb des gezeichneten Rechtecks eingefügt wird (vgl. Listing 4.2).

```
1  this.group.append("image") // Image-attributes of
   module
2  .attr("href", this.image)
3  .attr("height", this.height)
4  .attr("width", this.width)
5  .attr("id", this.id)
```

Listing 4.2: Image-Attribut beim Erstellen des SVG-Elements

Neben der angepassten Klasse `Module` beinhaltet die Datei `modules.js` weitere Unterklassen, die die Eigenschaften der Klasse erben, wie in Abbildung 4.2 veranschaulicht wird. Die Klassen stellen die unterschiedlichen Module (Generator, Schaltung und Oszilloskop). Diese enthalten einen eigenen Konstruktor, welcher die Parameter `x` und `y` erwartet und durch `super(x, y)`; die Konstruktorfunktion der Elternklasse aufruft. Außerdem beinhaltet die Konstruktorfunktion der jeweiligen Module eine Höhe und Breite sowie die Anzahl der Ein- und Ausgänge. Bei der Ausführung der Konstruktorfunktion werden des Weiteren vier Funktionen der Elternklasse aufgerufen, um das jeweilige Modul zu erzeugen, die Positionen der Ein- und Ausgänge zu berechnen und in die Liste der erstellen Module hinzuzufügen. Ein Beispiel für eine Konstruktorfunktion der Unterklasse `oscilloscope` wird in Listing 4.3 dargestellt.

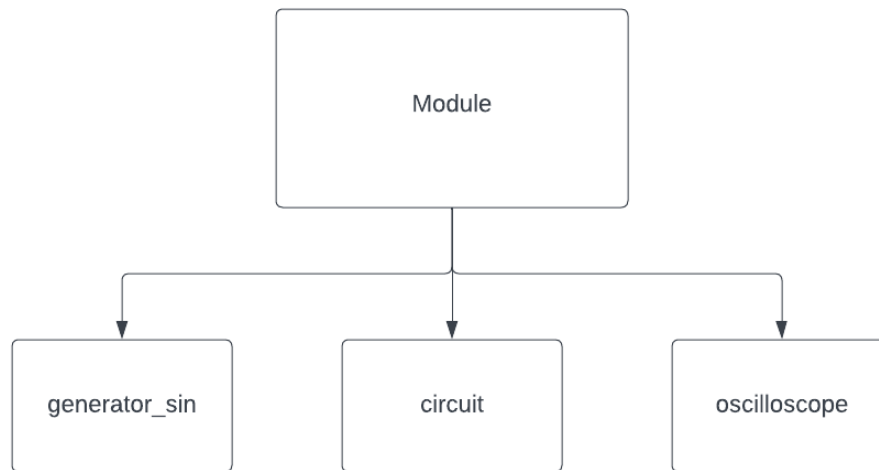


Abbildung 4.2: Struktur der Klassen der Funktionsbausteine des Verdrahtungseditors

```
1  class oscilloscope extends Module {
2      constructor(x, y)
3      {
4          super(x, y);
5          this.maxInputCount = 1;
6          this.maxOutputCount = 0;
7          this.width = 150;
8          this.height = 150;
9          this.image = "../Circuits/oscilloscope.png"
10
11
12          this.calcOutputOffset();
13          this.calcInputOffset();
14          this.build("rect");
15          this.addsInputToArray();
16      }
17 }
```

Listing 4.3: Konstruktorfunktion der Unterklasse oscilloscope

Die zugehörigen Bilder, welche in die SVG-Elemente beim Erstellen eingefügt werden, sind ebenfalls im Konstruktor der Module definiert und werden im Ordner `Circuits` als Portable Network Graphics (PNG)-Datei abgelegt. Für die bessere Unterscheidung der einzelnen Module wurden hier bereits einige Beispieldateien abgelegt.

4.3 Menüleiste und Einbindung der Simulation

Um die Webanwendung benutzerfreundlich und übersichtlich zu gestalten, wird die Menüleiste des Logikeditors wiederverwendet und entsprechend angepasst. Die Abbildung 4.3 zeigt die einzelnen Schaltflächen der Menüleiste für den Verdrahtungseditor.

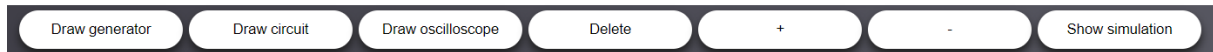


Abbildung 4.3: Schaltflächen der Menüleiste des CircuitJS-Verdrahtungseditors

Hier wurden neue Schaltflächen hinzugefügt, um die verschiedenen Bausteine für den Aufbau der elektrischen Schaltungen zu erstellen. Es gibt jeweils eine Schaltfläche für das Zeichnen eines Generators, einer Schaltung sowie eines Oszilloskops. Diese wurden in der `index.html`-Datei innerhalb des `<div>`-Elements der Menüleiste als `<button>`-Elemente implementiert, wie in Listing 4.4 zu erkennen ist. Mit dem Eventhandler `onclick` wird bei einer Interaktion mit der Schaltfläche die Funktion `place()` aufgerufen und ein neues Modul der jeweiligen Klasse (vgl. Kapitel 4.2) an den entsprechenden x- und y-Koordinaten erstellt.

```

1  <div class = "header" id="navigationBar" name="navigationBar">
2  <button class="moduleButton" onclick="place(new generator_sin(100,50))">Draw
   generator</button>
3  <button class="moduleButton" onclick="place(new circuit(500,50))">Draw
   circuit</button>
4  <button class="moduleButton" onclick="place(new oscilloscope(900,50))">Draw
   oscilloscope</button>

```

Listing 4.4: Implementierung der Schaltflächen in die Menüleiste (1)

Die Schaltflächen `Delete`, welche die Funktion um einen Baustein zu löschen aktiviert, sowie die Schaltflächen `+` und `-` für den Zoom des Simulationsfenster wurden aus dem Projekt des Logikeditors wiederverwendet. Des Weiteren wurde eine zusätzliche Schaltfläche `Show Simulation` erstellt, welche das `iFrame`-Element der CircuitJS-Simulation bei Betätigung der Schaltfläche einblendet bzw. ausblendet. Hierzu wurde die Funktion aus Listing 3.2 verwendet, welche den Style des `iFrame`-Elements ändert (vgl. Listing 4.5).

```

1  <input type="checkbox" id="deleteCheckbox" onchange="toggleDelete()"/>
2  <label class="moduleButton" id="deleteButton" for="deleteCheckbox">Delete</
   label>
3  <input class="moduleButton" type="button" value="+" onclick="zoomIn()"/>
4  <input class="moduleButton" type="button" value="-" onclick="zoomOut()"/>
5  <button class="moduleButton" onclick="showSimulation()">Show simulation</
   button>

```

Listing 4.5: Implementierung der Schaltflächen in die Menüleiste (2)

Um eine Interaktion mit der CircuitJS-Simulation während der Verwendung des Verdrahtungseditors zu ermöglichen, wurde das `<div>`-Element mit der ID `simulation`, in der sich das `iFrame`-Element der Simulation befindet, in das `<body>`-Element der `index.html`-Datei implementiert. Die Simulation wird am oberen Rand der Anwendung abgebildet, sodass diese durch die jeweilige Schaltfläche ausblendet werden kann und die Bedienung des Verdrahtungseditors auch bei eingeblendeter Simulation erfolgen kann.

4.4 Kontextmenü

Für die Interaktion mit den platzierten Modulen wurde das Kontextmenü des Logikeditors implementiert und entsprechend angepasst. Dieses Kontextmenü wird bei einem Rechtsklick auf den jeweiligen Baustein durch die Funktion `menuLogic(element)` erstellt. Hier wird der Funktion das jeweilige Modul übergeben. Um den verschiedenen Bausteinen des Verdrahtungseditors unterschiedliche Kontextmenüs zuzuweisen, wird zunächst der Parameter `classname` des Elements durch eine `if`-Anweisung abgefragt und das Kontextmenü entsprechend mit unterschiedlichen Schaltflächen beschrieben. Listing 4.6 zeigt die Erstellung des Kontextmenüs, wenn es sich bei dem übergebenen Element um einen Generator handelt.

```
1  if(element.classname == "generator_sin") {
2    var menu =
3      [
4        {
5          title: "properties",
6          action: function () {
7            let overlay = new Overlay();
8            console.log(document.querySelector('#left'));
9            overlay.render(document.querySelector('#left'));
10           overlay.show([[ "number", "Frequency", 100, 0, 10000 ], [ "number", "
Amplitude", 1, 0, 10 ]]);
11         },
12       },
13       {
14         divider: true,
15       },
16       {
17         title: 'delete',
18         action: function () {
19           console.log(element);
20           element.delete();
21         }
22       }
23     ];
24  }
```

Listing 4.6: Array für das Kontextmenü eines Generators

Für den Generator werden zwei Schaltflächen in einzelnen Objekten des Arrays `menu` erstellt. Ein weiteres Objekt mit dem Parameter `divider` trennt die beiden Schaltflächen mit einer horizontalen Linie. Bei der oberen Schaltfläche des erstellen Kontextmenüs wird bei Interaktion eine Funktion aufgerufen und ein neues Objekt der Klasse `Overlay` erstellt. Auf die Erstellung des Overlays wird in Kapitel 4.5 genauer eingegangen. Die Schaltfläche mit dem Titel `delete` löscht das ausgewählte Modul durch den Funktionsaufruf `element.delete()`. Der Array `menu` wird anschließend der Funktion des d3-context-menu Plugins übergeben und somit das Kontextmenü erstellt (vgl. Listing 4.7).

```
1 d3.contextMenu(menu)();
```

Listing 4.7: Funktionsaufruf des d3-context-menu Plugins

4.5 Overlay zur Konfiguration der Bausteine

Um die verschiedenen Module für den Verdrahtungseditor individuell anpassbar zu gestalten, müssen diese mit Parametern konfiguriert werden können. Hierzu dient ein entsprechendes Overlay, welches durch das Kontextmenü des jeweiligen Bausteins aufgerufen wird. Dabei wird ein neues Objekt `overlay` erstellt und der Klasse `Overlay` zugewiesen. Für dieses Objekt wird im nächsten Schritt die Funktion `overlay.render(document.querySelector('#left'))` aufgerufen. Diese Funktion implementiert zunächst das zugehörige CSS-Stylesheet `overlay.css` für das erstellte Overlay im `<head>`-Element der Dokuments. Der Parameter `parentSelector` dient dazu, das Overlay-Element an einer bestimmten Stelle auf der Webseite zu platzieren (vgl. Listing 4.8).

```
1 render(parentselector){
2     this.parentSelector = parentselector;
3     $('head').append('<link rel="stylesheet" href=".\\components\\overlay\\
overlay.css" />');
4 }
```

Listing 4.8: Render-Funktion der Klasse Overlay

Im nächsten Schritt wird die Methode `show()` aufgerufen, die zweidimensionales Array als Argument annimmt. Jedes Array enthält Informationen über eine Eingabe, die im Overlay angezeigt werden soll. Im Beispiel aus Listing 4.9 gibt es zwei Eingabefelder für die Konfiguration eines Generators: Ein numerisches Feld mit dem Label `Frequency` und ein weiteres numerisches Feld mit dem Label `Amplitude`. In diesen Eingabefeldern können die Frequenz und die Amplitude, für die Eingangsspannung der CircuitJS-Simulation gesetzt werden. Des Weiteren beinhalten die Arrays Attribute, um eine Schrittwerte sowie einen Minimal- und einen Maximalwert der Eingabe festzulegen.

```
1 overlay.show([[ "number", "Frequency", 100, 0, 10000 ], [ "number", "Amplitude", 1, 0, 10 ]]);
```

Listing 4.9: Funktionsaufruf der Klasse Overlay zum Erstellen der Eingabefenster

Die `show()`-Methode überprüft zunächst, ob bereits ein Overlay-Element auf der Seite vorhanden ist. Wenn nicht, wird ein neues Overlay-Element in das Dokument eingefügt, indem die `$.load()`-Methode von jQuery aufgerufen wird. Sobald das Element geladen ist, werden die Eingabefelder und eine Bestätigungsschaltfläche dynamisch erstellt und dem Overlay hinzugefügt. Jedes Eingabefeld wird als neues `<div>`-Element mit einem Label und einem Formularfeld erzeugt. Die Bestätigungsschaltfläche wird als `<button>`-Element am unteren Rand des Overlays hinzugefügt. Wenn ein Overlay-Element bereits auf der Seite vorhanden ist, wird dies sichtbar gemacht, indem dem Overlay die Klasse `visible` zugewiesen wird.

```
1 show(values){
2   this.overlay = $('<div>.overlay');
3   if(this.overlay.length === 0){
4     $(this.parentSelector).load('<div>.\components\overlay\overlay.html',
5     ,()=>{
6       this.overlay = $('<div>.overlay')[0];
7       this.form = $('<div>.form-content')[0];
8       this.overlay.classList.add("transition");
9       this.form.classList.add("transition");
10
11       values.forEach(value => {
12         this.createInput(value, $(this.form));
13       });
14       this.createButton(values, $(this.form));
15       setTimeout(() => $('<div>.transition').addClass('visible'), 1);
16     });
17   }
18   else{
19     setTimeout(() => $('<div>.transition').addClass('visible'), 1);
20   }
21 }
```

Listing 4.10: Show-Funktion der Klasse Overlay

Die Funktion `createInput` dient dazu, ein Eingabefeld für jeden übergebenen Wert zu erstellen. Der Parameter `value` ist ein Array, das die Informationen über das Eingabefeld enthält, und unterschiedliche Formate für numerische Eingabefelder oder Auswahlfelder hat. Für die einzelnen Eingabefenster wird ein `<div>`-Element erstellt, das aus einem `input`-Element, das das Feld beschriftet, sowie dem Eingabefeld besteht. Für die Erstellung des Eingabefelds wird durch eine `if`-Anweisung abgefragt, um welchen Typ es sich handelt. Handelt es sich um ein numerisches Eingabefeld, wird ein neues HTML-Element `<input>` mit den entsprechenden Eigenschaften erstellt. Bei einem Auswahlfeld wird im Gegensatz dazu ein `<option>`-Element mit den einzelnen

Auswahloptionen hinzugefügt (vgl. Listing 4.11).

```
1   var options = '<option>--- choose circuit ---</option>';
2   for(var i = 0; i < values[2].length; i++) {
3       options += '<option>' + values[2][i] + '</option>';
4   }
```

Listing 4.11: Erstellen des HTML-Elements für das Auswahlfeld

Die einzelnen Eingabefelder werden im Overlay erstellt, indem sie dem Array `inputField` hinzugefügt werden.

Die Funktion `createButton` erstellt die Bestätigungsschaltfläche des Overlays, um die eingegebenen Daten zu verarbeiten und das Overlay auszublenden. Der Funktionsparameter `values` wird verwendet, um festzustellen, welche Aktion ausgeführt werden soll, wenn die Schaltfläche geklickt wird. Handelt es um die Eingabefenster zur Konfiguration des Generators, wird eine neue Instanz von `ModuleManager` erstellt und die Funktion `updateModuleList` aufgerufen, um die festgelegte Frequenz und Amplitude an das Modul und der Simulation zu übergeben, wie in Listing 4.12 zu erkennen ist.

```
1   if(values[0][0] == "number") {
2       e.preventDefault();
3       this.simParam = [];
4       this.inputField.forEach((input)=>{
5           this.simParam.push(input.value);
6       })
7       new ModuleManager().updateModuleList(this.simParam[0], this.simParam[1])
8   ;
9   }
```

Listing 4.12: Setzen der festgelegten Frequenz und Amplitude des Generators

Wenn das Eingabefenster zum Festlegen einer Schaltung des Moduls `circuit` ist, wird das `Image`-Element für die ausgewählte Schaltung entsprechend aktualisiert und die passende Simulationdatei aus dem Ordner `war` in die CircuitJS-Simulation geladen (vgl. Listing 4.13).

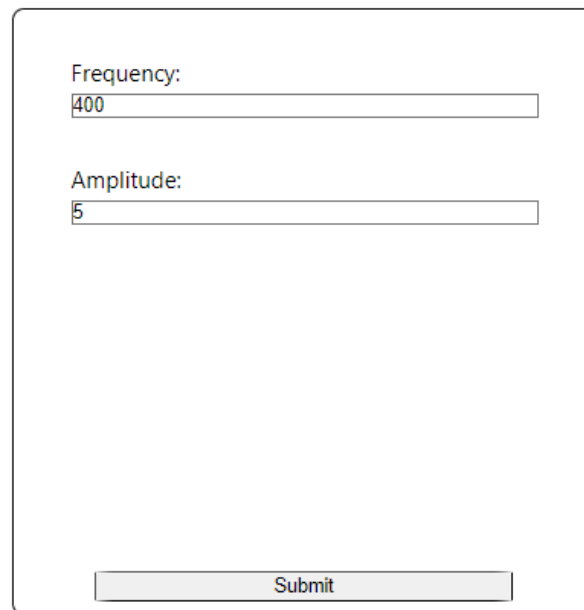
```
1   else if(values[0][0] == "select") {
2       e.preventDefault();
3       var moduleList = new ModuleManager().getModuleList();
4       this.newiFrame = document.getElementById("circuitFrame");
5       var element = moduleList[1];
6
7       // set simulation source and circuit image
8       if(this.inputField[0].value == "Lowpass") {
9           element.image = "../Circuits/LR_Tiefpass.png";
10          this.newiFrame.src = "../war/circuitjs.html?startCircuit=_lowpass_ext.txt";
11      }
```

```
11     }
12     else if(this.inputField[0].value == "Highpass") {
13         element.image = "./Circuits/CR_Hochpass.png";
14         this.newiFrame.src = "../war/circuitjs.html?startCircuit=
_highpass_ext.txt";
15     }
16     else if(this.inputField[0].value == "Resistor") {
17         element.image = "./Circuits/Resistor.png";
18         this.newiFrame.src = "../war/circuitjs.html?startCircuit=
_resistor_ext.txt";
19     }
20
21     // reload simulation
22     var divEl = document.getElementById("sim");
23     divEl.replaceChild(this.newiFrame, divEl.firstChild);
24     this.newiFrame.contentWindow.onsimloaded = simLoaded;
25
26     //Set new image
27     document.getElementById("1").setAttribute("href", element.image);
28 }
```

Listing 4.13: Setzen der ausgewählten Schaltung durch if-Abfrage

Nach der Verarbeitung der eingegebenen Daten aus den Eingabefenster wird das Overlay durch den Funktionsaufruf `this.hide()` geschlossen.

Die Abbildung 4.4 zeigt die Konfiguration eines Generators im erstellten Overlay. Hier befinden sich in der oberen Hälfte des Overlays die erstellten Eingabefenster, welche durch den Funktionsaufruf `overlay.show()` erstellt werden (vgl. Listing 4.9). Am unteren Rand des Overlays aus Abbildung 4.4 ist die Bestätigungsschaltfläche, welche die eingegebenen Daten verarbeitet und das Overlay schließt.



The image shows a web form with two input fields. The first field is labeled 'Frequency:' and contains the value '400'. The second field is labeled 'Amplitude:' and contains the value '5'. Below these fields is a 'Submit' button.

Abbildung 4.4: Eingabe der Frequenz und Amplitude für einen Generator-Baustein

4.6 Anpassung der CircuitJS-Simulation

Für die Anwendung des entwickelten Verdrahtungseditors wurden zunächst drei elektrische Schaltungen ausgewählt, die mit der CircuitJS-Simulation simuliert und über das Overlay ausgewählt werden können. Dies ist zum einen eine einfache Widerstandsschaltung sowie eine Hochpass- und eine Tiefpassschaltung. Damit die eingestellte Schaltung im Verdrahtungseditor schnell ersichtlich ist, sind hier entsprechende PNG-Dateien im Ordner `Circuits` hinterlegt, welche bei Auswahl der Schaltung angezeigt werden (vgl. Kapitel 4.2 und 4.5). Für diese Schaltungen wurden entsprechende CircuitJS-Simulationsdateien erstellt und im Ordnerverzeichnis `war` abgelegt.

Beim Laden der Webseite wurde als Standardschaltung die Hochpassschaltung im `iFrame`-Element der `index.html`-Datei festgelegt. Die Simulation wird mit dem Laden der Webseite gestartet, jedoch liegt an der Simulationsschaltung zu Beginn keine Eingangsspannung an, da die Frequenz und die Amplitude nicht gesetzt sind. In der Funktion `didUpdate(sim)` der `simLoad.js`-Datei wird zyklisch die Frequenz und die Amplitude der Eingangsspannung ermittelt. Hierfür dienen mehrere, ineinander verzweigte `if`-Abfragen. In Listing 4.14 wird zunächst die Liste der bereits platzierten Module abgefragt und in der Variablen `moduleList` abgespeichert. Im nächsten Schritt wird ermittelt, ob Baustein der Klasse `Generator` platziert wurde. Gibt es einen Generatormodul, wird abgefragt, ob dieser eine `output`-Verbindung zu einem anderen Modul hat. Ist eine Verbindung vorhanden, wird die Frequenz und die Amplitude des Generators aus `moduleList` abgefragt und der Simulation übergeben.


```
1  var moduleList = new ModuleManager().getModuleList();
2
3  if(moduleList[0] !== undefined) {
4
5      if(moduleList[0].classname == "generator_sin") {
6
7          if(moduleList[0].output.length == 0) {
8              freq = 0;
9              ampl = 0;
10         }
11
12         else if (moduleList[0].output[0].output !== 0) {
13             freq = moduleList[0].frequency;
14             ampl = moduleList[0].amplitude;
15         }
16     }
17 }
```

Listing 4.14: Setzen der Eingangsspannung der CircuitJS-Simulation

Beim Platzieren des Generators wird diesem zunächst eine Frequenz und eine Amplitude von 0 zugewiesen. Damit eine Eingangsspannung in der CircuitJS-Simulation gesetzt wird, muss also zum einen der Generator mit einem anderen Modul verbunden sein und zum anderen eine, über das Overlay gesetzte, Frequenz und Amplitude enthalten.

Um die Schaltung während der Simulation zu ändern, wird sie beim Bestätigen der Eingabe im Overlay durch den Funktionsaufruf aus Listing 4.15 neu geladen. Dabei wird die Funktion `simLoaded` aufgerufen, die die CircuitJS-Simulation neu lädt und wiederum die Callback-Funktionen zyklisch aufruft.

```
1  // reload simulation
2  var divEl = document.getElementById("sim");
3  divEl.replaceChild(this.newiFrame, divEl.firstElementChild);
4  this.newiFrame.contentWindow.onsimloaded = simLoaded;
```

Listing 4.15: Laden der Simulation mit neuer Simulationsdatei

5 Fazit und Ausblick

Das entwickelte Projekt des Verdrahtungseditors ermöglicht dem Benutzer das Erstellen und Simulieren von einfachen elektrischen Schaltungen in einer benutzerfreundlichen Weise. Dies bietet Möglichkeiten zur Integration des Editors in verschiedene Grundlagenvorlesung der Elektrotechnik.

Der Editor wurde auf Basis eines bestehenden Logikeditors sowie einer CircuitJS-Simulation aufgebaut. Hierfür wurden alle zu bearbeitenden Funktionen erfolgreich umgesetzt und getestet, wie beispielsweise der Implementierung und Anpassung der CircuitJS-Simulation sowie die Umsetzung neuer Baustein-Klassen. Des Weiteren bietet das Overlay zur Konfiguration der einzelnen Module eine gute Basis zur Weiterentwicklung, um gegebenenfalls weitere Konfigurationsmöglichkeiten hinzuzufügen.

Neben den umgesetzten Funktionalitäten gibt es innerhalb dieses Projektes Optimierungsmöglichkeiten in einigen Bereichen, welche für die zukünftige Bearbeitung der Anwendung sinnvoll sind. Hierzu zählt zum einen das automatische Suchen der Bausteine in der Modulliste. Für die vollständig funktionierende Anwendung des Editors muss aktuell zunächst ein Generator und anschließend eine Schaltung platziert werden. Dies könnte durch eine zusätzliche Schleifenabfrage gelöst werden, welche die Modulliste durchläuft, beim ersten Modul der Klasse Generator stoppt und diesen Eintrag weiterverarbeitet.

Des Weiteren bietet der aktuelle Stand des Projektes eine gute Ausgangsbasis, neue Funktionalitäten zu implementieren. Um eine genaue Analyse des Signals zu ermöglichen, kann in einer weiterführenden Arbeit ein virtuelles Oszilloskop implementiert werden, welches sowohl die Eingangsspannung des Generators als auch die Ausgangsspannung der Schaltung auf einem Oszilloskop-Bildschirm darstellt. Hierfür wurde bereits das Kontextmenü für den Baustein des Oszilloskops erstellt. Bei einer Interaktion mit der Schaltfläche `open oscilloscope` kann beispielsweise das Oszilloskop am unteren Ende der Webseite durch ein zusätzliches Canvas-Element dargestellt werden.

Außerdem besteht die Möglichkeit, den Verdrahtungseditor zu erweitern, sodass mehrere Schaltungen gleichzeitig simuliert und analysiert werden können, um somit den Lernerfolg von Studierenden in Grundlagenvorlesungen der Elektrotechnik durch interaktives Lernen zu steigern.

Literaturverzeichnis

- [1] *DHBW Mannheim: Elektrotechnik - Projekte des Studiengangs*, 7.03.2023. Adresse: <https://www.mannheim.dhbw.de/dual-studieren/bachelor/technik/elektrotechnik/projekte>.
- [2] J. Epperlein, *Javascript: Erweiterung eines Logikeditors für die Digitaltechnikvorlesung*, Mannheim, 2023.
- [3] V. Mostowoj, *Programmierung eines Virtuellen Oszilloskops mit HTML5 Canvas*, Mannheim, 2023.
- [4] T. Kahlert, *Visual Studio Code (1): Die Grundlagen*, 28.02.2023. Adresse: <https://www.microsoft.com/de-de/techwiese/blog/visual-studio-code-01-die-grundlagen.aspx>.
- [5] Microsoft, *Documentation for Visual Studio Code*, 28.02.2023. Adresse: <https://code.visualstudio.com/docs>.
- [6] P. Bühler, P. Schlaich und D. Sinner, *HTML und CSS: Semantik - Design - Responsive Layouts* (Bibliothek der Mediengestaltung), 2nd ed. 2023. Berlin, Heidelberg: Springer Berlin Heidelberg und Imprint Springer Vieweg, 2023, ISBN: 978-3-662-66663-0. DOI: 10.1007/978-3-662-66663-0.
- [7] G. Pomaska, *Webseiten-Programmierung*. Wiesbaden: Springer Fachmedien Wiesbaden, 2012, ISBN: 978-3-8348-2484-4. DOI: 10.1007/978-3-8348-2485-1.
- [8] *CSS-Vorteile und Grenzen*, Berlin, 2005. Adresse: http://page.mi.fu-berlin.de/mbudde/css_kurs/vorteile.html.
- [9] *d3-Kontextmenü - npm*, 10.04.2023. Adresse: <https://www.npmjs.com/package/d3-context-menu>.
- [10] J. Wolf, *HTML und CSS: Das umfassende Handbuch* (Rheinwerk Computing), 4., aktualisierte und überarbeitete Auflage. Bonn: Rheinwerk Verlag, 2021, ISBN: 9783836281171.
- [11] *HTML/Elemente/canvas - SELFHTML-Wiki*, 11.04.2023. Adresse: <https://wiki.selfhtml.org/wiki/HTML/Elemente/canvas>.
- [12] P. Pedamkar, „SVG vs Canvas“, *EDUCBA*, 31.10.2018. Adresse: <https://www.educba.com/svg-vs-canvas/>.
- [13] *HTML/Elemente/iframe - SELFHTML-Wiki*, 11.04.2023. Adresse: <https://wiki.selfhtml.org/wiki/HTML/Elemente/iframe>.