ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Entwicklung einer Internationalisierungslösung für das Content Management System Scrivito inklusive Evaluierung des Systems und Anwendung an einer internationalen Webseite.

vorgelegt von
Julian Krieger

betreut und begutachtet von
Prof. Dr. Thomas Kretschmer

Saarbrücken, 30. 09 2020

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 30. 09 2020

Julian Krieger

Zusammenfassung

Die Umsetzung von Webseiten und -anwendungen im internationalen Raum benötigt ein Content-Management-System, welches in der Lage, ist den einzigartigen Anforderungen einer internationalen Webseite gerecht zu werden. Die Landschaft der verfügbaren Content-Management-Systeme ist ständigen Änderungen ausgesetzt.

Die Qualität der Neulinge in dieser Branche lässt sich oft schlecht ohne einen Testeinsatz feststellen. Ihre Vor- und Nachteile müssen sorgfältig gegeneinander abgewogen werden.

In dieser Ausarbeitung wird daher geprüft, inwiefern die Vorteile von Scrivito-CMS dem Nachteil des Abhandenseins einer Internationalisierungsfunktion gegenüberstehen und ob sich dieser Nachteil beheben lässt. In dieser Ausarbeitung wird ein Konzept vorgestellt, dass eine solche Internationalisierungslösung präsentiert.

Im Rahmen der Erstellung eines Proof-of-Concepts einer Internationalisierungslösung und einer für Ergosign relevanten Webseite kann das Ergebnis dann evaluiert werden. Diese Evaluierung ist für die mögliche Wiederverwendung von Scrivito essentiell.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich in der Ausarbeitung dieser Thesis unterstützt und motiviert haben.

Zuerst gebührt mein Dank meinem betrieblichen Betreuer [redacted] der mir stets mit seiner hervorragenden Expertise, konstruktiver Kritik und hilfreichen Anmerkungen zur Seite stand.

Ich möchte mich außerdem bei meinem betreuenden Professor, [redacted], bedanken. Er führte mich sehr gut an die Vorgehensweise zur Bearbeitung einer Wissenschaftlichen Arbeit heran.

Weiterhin möchte ich noch [redacted] danken, welche mich in der Zeit der Bearbeitung stets motivierten und es immer schafften, meinen Stresspegel zu senken.

Ein besonderer Dank gilt vor allem [redacted], ohne deren Korrektur es mir nicht möglich gewesen wäre, eine Ausarbeitung in dieser Form anzufertigen.

Abschließend danke ich noch meiner Lerngruppe, bestehend aus meinen Kommilitonen und guten Freunden [redacted] für ihre großartige Unterstützung während des gesamten Studiums. Kaum eine Klausur wurde ohne sie bestritten, kein Arbeitsblatt ohne sie bearbeitet und in keiner Projektgruppe konnten sie fehlen. Ohne sie wäre ich nicht in der Lage gewesen, mein Studium in dieser Form und Qualität abzuschließen. Ich wünsche ihnen nichts all das beste für ihre zukünftige Laufbahn.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Ergosign ist ein multinationales Unternehmen mit verschiedenen Unternehmenspartnern. Zu diesen zählt unter anderem das chinesische Design-Unternehmen „Artop Group“. Mit diesem gründete Ergosign 2019 das Joint-Venture „[redacted]“. [redacted] ist eine User-Experience Design-Agentur, die ihren Hauptsitz im chinesischen Chongqing hat. Das Joint-Venture legt seinen Fokus hauptsächlich auf den asiatischen Markt. [30]

Um internationale Unternehmen im Internet erfolgreich zu repräsentieren, muss deren Internetpräsenz durch Ergosign mit multinationalen Funktionen ausgestattet werden. Wünscht ein Kunde weiterhin, seine Inhalte über ein Content-Management-System verwalten zu können, muss durch Ergosign ein CMS bereitgestellt werden, welches über internationale Funktionen verfügt. Zu diesen gehören unter anderem dass Verwalten von internationalen Inhalten, die Möglichkeit, neue Inhalte einzupflegen und zu verwalten und eine Benutzeroberfläche, die diese Verwaltung einfach ermöglicht. Da sich die Auswahl an Content-Management-Systemen stetig erweitert und Ergosign die besten Lösungen für Kunden bereitstellen möchte, müssen neue Content-Management-Systeme auf Kundenspezifische Anforderungen getestet werden. Ergosign stellt weiterhin die Anforderung an ein Content-Management-System, Designvorstellungen Pixel-perfekt umsetzen zu können.

Neben der Unternehmenspartnerschaft mit der Artop-Group arbeitet Ergosign auch mit dem in Berlin ansässigen Unternehmen „Infopark“ zusammen [41]. Infopark bietet unter anderem das internationale Content-Management-System „Scrivito“an. Scrivito ist ein Cloud-basiertes Content-Management-System für Enterprise-Kunden und ist vor allem aufgrund seines Aufbaus auf einem modernen Web-Paradigma für Ergosign wirtschaftlich interessant.

Innerhalb dieser Bachelorthesis wird die Umsetzung einer internationalen Webseite innerhalb von Scrivito bearbeitet, um für Ergosign einen Überblick über die Leistungsfähigkeit, Funktionsweise und Qualität von Scrivito zu schaffen. Diese Faktoren können nach der Fertigstellung dieser Ausarbeitung von Ergosign bewertet werden. Anschließend kann Scrivito mit anderen Content-Management-System verglichen und eventuell für zukünftige Projekte wiederverwendet werden.

1.2 Aufgabenstellung und Zielsetzung

Ziel der Ausarbeitung ist es, Scrivito anhand eines konkreten Anwendungsfalls zu testen. Diesen liefert das mit Artop-Group gegründete, chinesische Joint-Venture [redacted], [redacted] benötigt eine Webseite, die Informationen zum Unternehmen und Kontaktmöglichkeiten bereitstellt. Scrivito liefert [redacted] die Möglichkeit, die auf der Webseite verfügbaren Informationen ohne zusätzlichen Programmieraufwand anzupassen.

Da Scrivito noch nicht über eine Internationalisierungsfunktion verfügt, soll diese im Rahmen der Ausarbeitung ebenfalls entwickelt und eingesetzt werden. Anschließend an die Bachelorthesis werden die Inhalte durch chinesische Korrespondenten von Ergosign übersetzt und in das Content-Management-System eingepflegt.

Die Implementierungen der [redacted]-Webseite und der Internationalisierungsfunktion für Scrivito sind innerhalb dieser Bachelorthesis zu gleichen Teilen gewichtet. Es wird geprüft, inwieweit Scrivito zur Umsetzung in der Lage ist und welche Anforderungen dafür bearbeit werden müssen. Der Fokus liegt dabei auf der Planung der späteren Implementierungsarbeiten. Anschließend werden die Anforderungen in ein Konzept übertragen und schließlich innerhalb einer konkreten Lösung implementiert.

1.3 Aufbau der Artbeit

Diese Ausarbeitung wurde im Rahmen einer Bachelor-Thesis bei der Ergosign GmbH in Saarbrücken angefertigt. Die Betreuung seitens der Universität wurde von Prof. Dr. Thomas Kretschmer übernommen. Auf betrieblicher Seite übernahm Jan Zipfler die Aufsicht.

Nachdem in den oberen Teilen ein Überblick über das Thema der Arbeit und ihr Ziel erläutert wurde, werden in Kapitel 2 alle zur Verständnis der Arbeit nötigen Grundlagen geklärt. In diesem finden sich Grundkenntnisse über Technologien, die in der Webentwicklung, der Entwicklung innerhalb eines Content-Management-Systems oder zur Internationalisierung eines Software-Systems für diese Arbeit relevant sind. Es beinhaltet weiterhin wichtige Begriffe und Abkürzungen sowie deren Bedeutungen. Hier und in allen weiteren Kapitel befinden sich Quellenverweise, die einem Leser die Möglichkeit geben, die Informationsquelle einer Kernaussage ausfindig zu machen. Jedes Unterkapitel der Grundlagen ist so konzipiert, dass seine Konzepte auf die der vorangegangenen Unterkapitel aufbauen.

Kapitel 3 stellt zunächst für jedes der beiden Hauptziele des Themas die jeweilige Ausgangslage innerhalb einer sogenannten „Ist-Analyse“dar. Anschließend wird im Unterpunkt der „Soll-Analyse“ das Ziel und die zur Umsetzung nötigen Anforderungen des jeweiligen Themenpunktes festgestellt.
Kapitel 5 konzipiert eine Umsetzung der Anforderungen auf detaillierte Art und Weise.

Dazu werden die Anforderungen in elementare Teile zerlegt und erläutert. Durch sogenannte „Wireframes“ wird die Umsetzung grafisch dargestellt.

Das sechste Kapitel stell die für diese Thesis erarbeitete Implementierung vor, welche die Anforderungen und Ziele des Themas innerhalb eines „Proof-of-Concept“ umsetzt. Dort finden sich konkrete Quelltext-Beispiele, die dem Leser einen Überblick über die Funktionsweise und Implementierungsentscheidungen geben. Weiterhin wird er an die Entwicklung mit Scrivito und die dazu gehörige Internationalisierungslösung herangeführt.

Das siebte Kapitel evaluiert die die Umsetzung der [redacted]-Webseite und des Internationalisierungssystemes anhand eines gängigen Bewertungsschemas für Software-Systeme. Es geht unter anderem auf Geschwindigkeit, Qualität, Erweiterbarkeit und Wartbarkeit des erarbeiteten Systemes ein. Weiterhin wird analysiert, inwiefern die in Kapitel 3 dargestellten Anforderungen innerhalb des Zeitraumes umgesetzt sind, der zur Anfertigung der Bachelorthesis zur Verfügung stand.

Anschließend wird die Ausarbeitung zusammengefasst und ein Ausblick über mögliche, zukünftige Weiterarbeit an der Implementierung der Lösung geboten.

2 Grundlagen

2.1 CSS

„Cascading Style Sheets“ (CSS) ist eine Stylesheet-Sprache, welche zur Beschreibung des Erscheinungsbildes eines „Hypertext Markup Language“ (HTML)-Dokumentes verwendet wird [59]. Dazu können – unter anderem unmittelbar im Zieldokument oder in separaten „.css“-Dateien – Elemente mithilfe von sogenannten Selektoren ausgewählt und anschließend deren Erscheinungsbild mit unterschiedlichen Parametern konfiguriert werden [13, Kap. 2]. Ein solcher Selektor wählt eine beliebige Anzahl von Elementen anhand ihrer Attribute, Zustände oder Beziehungen zu umliegenden oder beinhalteten Elementen aus [13, Kap. 2] [5, Kap. 5]. Es ist möglich, mehrere Stylesheets innerhalb eines HTML-Dokumentes einzubinden. Diese geordnete Zusammenstellung von Stylesheets wird „Kaskade“ genannt [60, Kap. 1, Abs. 3]. Ein Internetbrowser ist dann in der Lage, alle in den Stylesheets vorhandenen Regeln zu einem einzigen Regelsatz zusammenzuführen. Dieser Prozess wird als „Kaskadierung“ bezeichnet [60, Kap. 1, Abs. 3]. Existieren mehrere Selektoren, die dasselbe Element auswählen, werden die Regeln des Selektors auf das Element angewandt, der die höhere „Spezifität“ aufweist [60, Kap. 3, Abs 2.2] [13, Kap. 9].

Obwohl CSS für gewöhnlich mit dem Styling von HTML-Dokumenten einer Webseite in Verbindung gebracht wird, kann es in jeder XML-basierten Sprache verwendet werden [42].

2.1.1 SASS

Das folgende Unterkapitel ist sinngemäß aus dem offiziellen „Syntactically Awesome Style Sheets“ (SASS)-Leitfaden entnommen [83].

SASS ist ein Superset von CSS, welches von Hampton Catlin und Nathan Weizenbaum 2006 veröffentlicht wurde [82]. Es erweitert Stylesheets um Funktionen, die in CSS nicht vorhanden sind. Damit SASS-Stylesheets in Browsern verwendet werden können, muss deren Inhalt vor der Einbindung in ein HTML-Dokument durch einen Preprozessor zu syntaktisch korrektem CSS umgewandelt werden. Neben der ursprünglichen, eingerückten Syntax verfügt SASS seit Version 3 über eine CSS-ähnliche Syntax mit geschweiften Klammern unter den Namen „Sassy CSS“ (SCSS) [84] [62].

SASS erlaubt es, CSS-Selektoren und -Regeln hierarchisch zu verschachteln und somit semantisch zu gruppieren. Das ermöglicht es, die Beziehungsstruktur von HTML-Elementen im CSS Code ungefähr abzubilden, was der Wartbarkeit und Lesbarkeit des

Quellcodes dient. Ist ein Selektor einem Eltern-Selektor untergeordnet, wird letzterer bei der Kompilierung zu CSS vor den ersteren gesetzt. Damit kann bei der Auswahl mehrerer, einem Eltern-Element untergeordneter Elemente die Wiederholung des Selektors auf das Eltern-Element verhindert werden (siehe Kapitel 6: 6.1 und 6.2). Weiterhin erlaubt SASS es dem Autor, sein Stylesheet in modulare Teile zu splitten und diese nur dort in andere Stylesheets einzubinden, wo sie benötigt werden. Darüber hinaus orientiert es sich durch Konzepte wie Vererbung und Komposition sogenannter „Mixins“ an einer objektorientierten Programmiersprache. Somit kann eine Redundanz von Regeln und Definitionen verhindert werden.

2.1.2 CSS Module

> *There are only two hard problems in Computer Science:*
> *cache invalidation and naming things – Phil Karton [55]*

Um mehrere HTML-Elemente auszuwählen und deren Aussehen anzupassen, kann man diese mit einem „Klassennamen“ versehen [110, Kap. 3, Abs. 3.6]. Dieser kann über CSS-„Class-Selectors“ ausgewählt und mit Regeln versehen werden [13, Kap. 6, Abs. 4]. Bei der Verwendung von Klassennamen und dazugehörigen Selektoren muss man jedoch darauf achten, dass ihre Namensgebung eindeutig ist, da sonst ungewollte Überschneidungen auftreten könnten [13, Kap. 9].

Um dieses Problem zu lösen und das Styling von Gruppen von HTML-Elementen komponentenweise voneinander abzugrenzen, ist es nötig, die Eindeutigkeit von CSS-Selektoren gewährleisten zu können [18]. Dazu existieren unterschiedliche Methoden. Unter anderem können die Namen der CSS-Klassen per Konvention gewählt werden. Nach der Vorgehensweise der „BEM“-Methodik entsteht ein CSS-Klassenname beispielsweise aus dem Namen seines umgebenden „Blocks“, gefolgt von einer Beschreibung des darzustellenden „Elementes“ und einem „Modifikator“ [69]. Aus einem Input-Feld mit entsprechendem Zustand eines umgebenden Form-Elementes entsteht dann beispielsweise der Klassenname `form__input--disabled` [69].

Um die Erzeugung eines eindeutigen Klassennamens programmatisch zu automatisieren, verwenden einige Werkzeugketten zum Erzeugen von JavaScript-Software „CSS-Module“ [17] (siehe 2.4.2). Laut Spezifikation kann eine CSS-Datei in Modulform innerhalb von JavaScript-Quellcode importiert und dort wie ein JavaScript-Objekt verwendet werden [17]. Die Attribute dieses JavaScript-Objekts stellen eine Zuordnung zu den Klassennamen des CSS-Moduls dar [17]. Anschließend können sie programmatisch auf das „class“-Attribut eines HTML-Elementes gelegt werden [17]. Verschiedene Implementierungen der CSS-Modules-Spezifikation wandeln abschließend im Bauprozess unterschiedlicher Werkzeugketten jeden so festgelegten CSS-Klassennamen in einen global eindeutigen um [112] [63].

2.2 JavaScript

Die folgenden Unterkapitel und Abschnitte sind sinngemäß angelehnt an das Buch „JavaScript: the first 20 years“ von Brendan Eich, dem Erfinder von JavaScript und Allen Wirfs-Brock, dem Editor der sechsten Version von ECMAScript [117].

Die Begriffe „JavaScript“ und „ECMAScript“ werden umgangssprachlich äquivalent verwendet. Korrekterweise ist JavaScript jedoch eine Programmiersprache, welche original unter dem Namen „Mocha“ bekannt und von Brendan Eich für den Browser „Netscape“ entwickelt wurde. Da der Programm-Code von JavaScript nicht quelloffen war, entwickelte Microsoft die Script-Sprache „JScript“ für den konkurrierenden Browser „Internet Explorer“. Da sich die Implementierungen von JavaScript und JScript unterschieden, war es Entwicklern von Webseiten nur schwer möglich, ohne Mehraufwand die Funktionalität ihrer Seite in einem Internet-Browser zu gewährleisten, der mit der jeweils anderen Script-Sprache arbeitete.

Um JavaScript und dessen Ableger zu standardisieren und übereinstimmende Implementierungen und Verhaltensweisen dieser zwischen allen Browser zu ermöglichen, standardisierten Mitarbeiter einiger Branchengrößen, unter ihnen Netscape und Microsoft, in Zusammenarbeit mit der Normungsorganisation „ECMA International“ eine Programmiersprache. Dieser Standard trägt den Namen „ECMA-262“ [45]. Die Entwicklung von ECMA-262 wird vom technischen Komitee „TC39“ überwacht. Der Name der daraus resultierenden, ECMA-262-konformen Programmiersprache, „ECMAScript“, ist ein Kompromiss der an der Entwicklung des Standards teilnehmenden Firmen und Organisationen.

2.2.1 ES6+

Laut einer Studie von „W3 Tech Surveys“ benutzen 96.4 Prozent aller Webseiten im Internet die Programmiersprache JavaScript [109]. Zudem arbeiten fast 70 Prozent aller hauptberuflichen Entwickler damit [97].

JavaScript erhielt anfangs nach seiner Veröffentlichung nur spärlich Neuerungen. So etwa liegen die Daten der Veröffentlichungen der Versionen „ECMAScript 3“ und „ECMAScript 5“ nach der eingestellten Arbeit an der vierten ECMAScript Version 10 Jahre auseinander. Die im Jahr 2015 veröffentlichte Version „ECMAScript 2015“ (auch ES6 genannt) war jedoch von großer Bedeutung geprägt. Sie war die erste der ECMAScript-Versionen, die fortan innerhalb eines jährlichen Zyklus veröffentlicht werden sollten. ES6 brachte neben diversen Funktionen wie Klassendefinitionen und asynchroner Programmierung auch die Möglichkeit mit sich, jeglichen Quellcode in sogenannte „Module“ aufzuteilen und innerhalb getrennter Dateien zu importieren und exportieren.

Nachfolgende „Proposals“ („Vorschläge“) von Funktionalitäten in neueren ECMAScript-Versionen folgen einem bestimmten Prozedere, welches vom TC39-Komitee überwacht wird. In diesen durchläuft das Proposal nacheinander fünf Stadien. Zum Zeitpunkt des ersten Stadiums wird der Vorschlag als sogenannter „Strawman“ ohne einen formalen

Prozess oder dedizierte Anforderungen von Mitgliedern des TC39-Komitees oder ECMA-International initiiert. Um den formalen Arbeitsprozess zu beginnen, muss anschließend ein „Champion“ identifiziert werden, der den Vorschlag durch die weiteren Phasen leitet. Danach muss das Proposal die Phasen „Proposal“, „Draft“ (Entwurf), „Candidate“ (Kandidat) und schließlich „Finished“ (Fertiggestellt) absolvieren, um eventuell in die nächste Veröffentlichung einer ECMAScript Version eingebaut werden zu können. Dabei muss der Abschluss eines jeden Schrittes vom TC39-Komitee akzeptiert werden. Die jeweils neueste Version des Entwurfs der ECMAScript-Spezifikation wird dabei umgangssprachlich „ESNext“ oder „ES.next“ genannt. Sie enthält Funktionen, die zwar durch das TC39-Komitee als „Fertiggestellt“ markiert sind, aber noch nicht veröffentlicht wurden.

2.2.2 JavaScript als dynamische Programmiersprache

Durch dynamische, schwache Typisierung besitzt JavaScript einige Funktionalitäten, die in statisch typisierten Programmiersprachen nicht vorhanden sind. Diese erschweren jedoch die statische Analyse von JavaScript-Quellcode. Chetan Conikee, Gründer und Erfinder der statischen Analyse-Platform „ShiftLeft“ erläutert einige Beispiele dazu in seinem Blogpost „An Oxymoron: Static Analysis of a Dynamic Language (Part 3)“.

Es ist beispielsweise möglich, einem JavaScript-Objekt nach dem Anlegen über seinen Konstruktor weitere Eigenschaften hinzuzufügen oder zu entziehen. Diese Möglichkeit birgt allerdings die Gefahr, dass die Struktur eines Objekts zur Laufzeit nicht fest definiert ist. Weiterhin ist es einem Programm zur statischen Analyse von Quellcode nicht oder nur schwer möglich, vorherzusagen, ob eine Variable an einem bestimmten Punkt zur Laufzeit die Werte `undefined` oder `null` annehmen kann. Bei einem Versuch, auf Parameter oder Eigenschaften eines Objekts zuzugreifen, welche nicht existieren, bricht die Ausführung des Programmcodes eventuell durch einen Laufzeitfehler ab. Die von JavaScript angewandte automatische Typenumwandlung bei der Nutzung eines Operators mit zwei Variablen verschiedener Typen kann ebenfalls zu unerwartetem Fehlverhalten führen (siehe Abbildung 2.1). [14]

Table 3-2. JavaScript type conversions

Value	Converted to:			
	String	Number	Boolean	Object
undefined	"undefined"	NaN	false	*throws TypeError*
null	"null"	0	false	*throws TypeError*
true	"true"	1		new Boolean(true)
false	"false"	0		new Boolean(false)
"" (empty string)		0	**false**	new String("")
"1.2" (nonempty, numeric)		1.2	true	new String("1.2")
"one" (nonempty, non-numeric)		NaN	true	new String("one")
0	"0"		**false**	new Number(0)
-0	"0"		**false**	new Number(-0)
NaN	"NaN"		**false**	new Number(NaN)
Infinity	"Infinity"		true	new Number(Infinity)
-Infinity	"-Infinity"		true	new Number(-Infinity)
1 (finite, non-zero)	"1"		true	new Number(1)
{} (any object)	*see §3.8.3*	*see §3.8.3*	true	
[] (empty array)	""	0	true	
[9] (1 numeric elt)	"9"	9	true	
['a'] (any other array)	*use join() method*	NaN	true	
function(){} (any function)	*see §3.8.3*	NaN	true	

Abbildung 2.1: JavaScript-Typenumwandlung [32, Kap. 3, Abs. 8]

2.2.3 TypeScript

Die im folgenden Abschnitt verwendeten Begriffe und Informationen stammen, sofern sie nicht anders markiert sind, aus dem offiziellen „TypeScript-Handbuch“ [44] und dem Artikel „TypeScript: JavaScript Development at Application Scale“ von Soma Somasegar (Microsoft) [96].

TypeScript ist ein typisiertes Superset für JavaScript von Microsoft [104]. Das bedeutet, dass jeglicher JavaScript-Quellcode auch gleichzeitig valider TypeScript Code ist. Durch statische Typisierung von JavaScript-Code mit zusätzlicher Syntax erlaubt TypeScript es, Code mit vom Compiler überprüfbaren Typen zu versehen und statisch analysieren zu lassen. Die Strenge dieser Überprüfung kann durch unterschiedliche Konfigurationsparameter zusätzlich angepasst werden. Durch TypeScript ist es möglich, die in 2.2.2 genannten potenziellen Fehlerquellen vor der Ausführung des Programms abzufangen. Dieser Vorgang vereinfacht die Entwicklung und Wartung bei Projekten, die eine große Menge an Quellcode oder eine Vielzahl an Entwicklern aufweisen.

Es ist möglich, eine Klasse oder eine Funktion mit einem Interface zu versehen, welches dann über die Typenüberprüfung von TypeScript eine Garantie über den Aufbau der Parameter und Rückgabewerte liefert. Diese bietet beim Lesen oder Nutzen von Quellcode eine Hilfestellung zum besseren Verständnis. Zudem ist es möglich, für Typen oder Interfaces festzulegen, ob diese oder enthaltene Typen zur Laufzeit nur optional vorhanden sind. Bei der Verwendung von Objekten vom jeweiligen Typ oder Interface muss diese Eventualität dann immer behandelt werden. TypeScript ist zudem über Typendefinitionen in der Lage, der Programmierumgebung eines Softwareentwicklers Hilfestellungen zur korrekten Benutzung der verwendeten Programmbibliothek zu Verfügung zu stellen.

Typendefinitionen können entweder unmittelbar neben dem Quellcode, oder in getrennten Deklarationsdateien mit der Endung „*.d.ts“ beschrieben werden. Letztere befinden sich mit einer JavaScript-Datei, sofern sie sich – bis auf die Dateiendung – den Namen teilen, in einer Dualbeziehung. Sie ähneln in ihrer Verwendung einer „Header-Datei“ der Programmiersprache *C*.

Einige JavaScript-Bibliotheken, welche eine Typendefinition ihrer Schnittstelle nicht direkt im Paket bereitstellen, besitzen eine solche im Open-Source-Repositorium „DefinitelyTyped“ [22].

Da TypeScript-Code in den meisten JavaScript-Laufzeitumgebungen nicht direkt ausführbar ist, muss er über einen TypeScript-Transcompiler übersetzt werden [43]. Bei diesem Schritt werden alle von TypeScript verwendeten Interfaces und Typendefinitionen sowie Typenanmerkungen aus dem generierten Quellcode entnommen, um syntaktisch korrekten JavaScript Code zu erzeugen. Andere Konstrukte wie „Enums“ werden im Kompilierungsvorgang durch JavaScript-konforme Elemente ersetzt [29].

2.2.4 Node.js

„Node.js“ ist eine quelloffene Laufzeitumgebung zur Ausführung von JavaScript-Code außerhalb eines Webbrowsers [71]. Dazu benutzt Node.js die von Google entwickelte JavaScript-Engine „V8“ [71]. Die von V8 implementierte JavaScript-Laufzeitumgebung läuft plattformübergreifend auf Windows und Unix-basierten Betriebssystemen [106]. Durch seine asynchrone, eventbasierte Architektur ist ein in Node.js entwickelter Server in der Lage, eine große Anzahl von Anfragen gleichzeitig zu verarbeiten [3]. Im Gegensatz zur Art und Weise, auf die andere serverseitige Programmiersprachen und Laufzeitumgebungen nebenläufige Programmierung umsetzen, verwendet Node.js kein Multithreading [3]. Stattdessen werden in Node.js beim Aufruf asynchroner Methoden sogenannte „Event-Handler“-Funktionen übergeben, die ausgelöst werden, wenn bestimmte Ereignisse erzeugt werden [3]. Ein Beispiel dafür ist eine JavaScript-Funktion, die bei der Anfrage eines Client auf den Server mit dem Satz „Hello, World“ antwortet. Durch eine solche „nicht-blockierender“ Funktion wird gewährleistet, dass der Programmablauf nicht durch die Ausführung langwieriger Funktionen gestoppt wird. Deswegen kann von diesen Funktionen unabhängiger Quelltext nebenläufig ausgeführt werden [72]. Trennt man nichtblockierende und blockierende Funktionen gänzlich voneinander, ist man je nach Situation in der Lage, Probleme wie „Race-Conditions“ oder „Dirty-Reads“, die in der nebenläufigen Programmierung auftreten können, zu vermeiden [72].

2.3 Webanwendungen

Der folgende Abschnitt stammt sinngemäß aus dem Blog-Post „What is a Web Application“ von Robert Gibb, einem Content-Marketing-Manager des US-amerikanischen Content-Delivery-Anbieters „StackPath“ [37].

Eine Webanwendung ist ein Computerprogramm, welche über einen Server verschiedene Dienste auf beliebig vielen Endgeräten bereitstellt. Sie funktioniert nach dem Client-Server-Modell und ist aus einer Kombination von Software der Client-Side- und Server-Side-Programmierung aufgebaut. Die durch serverseitige Software zur Verfügung gestellten Informationen werden dabei gegebenenfalls durch Skripte der Client-Side weiterverarbeitet und anschließend auf dem Endgerät eines Nutzers dargestellt. Das ermöglicht es unter anderem, allen Benutzern die gleiche Version einer Webanwendung zur Verfügung zu stellen. Anwender profitieren ebenfalls davon, dass die Webanwendung nicht auf ihrem Endgerät installiert werden muss.

In den folgenden Abschnitten werden einige Paradigmen vorgestellt, die in der Entwicklung einer Webanwendung relevant und für das Verständnis dieser Thesis notwendig sind.

2.3.1 Solution Stack

Bei einem „Solution Stack“ oder „Software Stack“ handelt es sich um eine Zusammenstellung verschiedener Software-Technologien, die miteinander interagieren und in ihrer Zusammenarbeit ein Softwareprodukt – in diesem Fall eine Webanwendung – vollständig abbilden [95].

Wichtig ist, dass es sich bei den Teilen eines Solution-Stack nicht etwa um generische Softwarekategorien handelt, sondern um eindeutige, festgelegte Softwareprodukte innerhalb ihrer Art. Stößt eine konkrete Zusammenstellung solcher Softwarekomponenten innerhalb der Branche auf Akzeptanz und/oder kann hohe Benutzerzahlen vorweisen, wird sie oft mit einem Namen oder einem Akronym aus den Anfangsbuchstaben der benutzten Bausteine versehen. [6]

Im Fall eines Stacks innerhalb der Webentwicklung legt dieser einzelne Produkte fest, die für das Betreiben einer Webanwendung notwendig sind [80]. Meist vertreten ist jeweils eine Lösung der Kategorien „Betriebssystem“, „Datenbanksoftware“, „Serversoftware“ und „Programmiersprache“ [80]. Die Auswahl eines Stacks beziehungsweise seiner Komponenten wird vor allem durch die Anforderungen an das zu entwickelnde Softwareprodukt beeinflusst [80].

2.3.2 Statische und Dynamische Webseiten

Eine statische Webseite ist eine Webseite, die von einem Web-Server unmittelbar ohne vorherige Aufbereitung an einen Anwender ausgeliefert wird [37]. Der Inhalt der Webseite wird allen Besuchern äquivalent angezeigt [66]. Rein statische Webseiten verfügen im Gegensatz zu dynamischen Webseiten nicht über eine Datenbank, Authentifizierungsalgorithmen oder ein komplexes Backend [7]. Ihr Inhalt muss oft manuell vom Betreiber der Webseite angepasst werden [66]. Im Gegensatz dazu stellen dynamische Webseiten bei unterschiedlichen Nutzern verschiedene Inhalte dar und reagieren auf deren Verhalten gegebenenfalls mit Inhaltsänderungen [37].

Es existiert keine klare Abgrenzung zwischen einer Webseite und einer Webanwendung, da beide Begriffe umgangssprachlich synonym verwendet werden. Eine Webanwendung wird jedoch oft subjektiv anhand ihrer Interaktivität von statischen, rein informativen Webseiten unterschieden [92]. Ein weiteres mögliches Unterscheidungsmerkmal ist das Ziel einer Webseite oder -anwendung. Während eine statische Webseite Informationen feststehende Informationen über ein Thema liefert [7], stellen dynamische Webanwendungen Dienstleistungen im Internet zur Verfügung, die ansonsten innerhalb einer Desktop-Anwendung existieren würden [37].

2.3.3 Server-Side-Programmierung

Das folgende Unterkapitel gibt Informationen aus dem Artikel „Introduction to the server side“ des „Mozilla Developer Network“ (MDN) sinngemäß wieder [52].

Unter „Server-Side-Programmierung“ versteht man eine Technik, die auf Interaktionen durch einen Nutzer bei der Navigation durch eine Webanwendung dynamisch reagiert. Ein konkreter Anwendungsfall besteht dabei aus der Generierung von dynamisch erstellten HTML-Dokumenten. So können einem Nutzer, wenn er mit der Web-Anwendung interagiert, speziell auf ihn zugeschnittene Inhalte angezeigt werden. Möchte der Client beispielsweise eine Unterseite aufrufen, wird, sobald er auf den entsprechenden Link klickt, eine Anfrage an den Server übertragen. Dieser antwortet mit einem HTML-Dokument, das exakt die abgefragte Unterseite darstellt. Das Gerät des Nutzers kann das entsprechende Dokument dann anzeigen.
Generiert werden die HTML-Dokumente unter anderem mit sogenannten „Templates“. Innerhalb eines Templates werden an den entsprechenden Stellen im Dokument Marker oder „Template-Parameter“ gesetzt, welche zur Laufzeit durch konkrete Daten ersetzt werden [34, p.147]. Die Software zum Ersetzen dieser Template-Parameter wird „Template-Engine“ genannt [100]. Weiterhin stellen einige Template-Engines die Funktionalität bereit, Programmierlogik in Form von „Scriptlets“ in ein Template einzubetten, welche dann zum Zeitpunkt der Generierung evaluiert werden [34].

2.3.4 Static-Site-Generatoren

Der folgende Abschnitt ist sinngemäß dem Artikel „What is a Static Site Generator? And 3 ways to find the best one“ von Phil Hawksworth entnommen. Phil Haksworth ist ein Mitarbeiter des in San Francisco, USA, ansässigen Cloud-Computing-Unternehmens „Netlify“ [70].

„Static-Site-Generatoren“ erlauben es, der Inflexibilität von statischen Webseiten entgegen zu wirken [66]. Im Gegensatz zu dynamischen Webseiten, welche jeweils ein neu generiertes HTML-Dokument aus Templates und Inhalt pro Unterseite auf Nachfrage liefern, berechnen Static-Site-Generatoren jede mögliche Seitenansicht im Voraus. Anschließend können die so erzeugten HTML-Dokumente und Inhalte auf einen Web-Server übertragen werden. An dieser Stelle werden vom Web-Server keine Änderungen mehr vorgenommen. Dieses Verfahren hat den Vorteil, das der Web-Server für die Abgabe von

Informationen an den Nutzer keine zusätzliche Rechenzeit mehr aufwenden muss. Im Unterschied zu dynamisch generierten Webseiten bleibt das Kompilieren von Templates pro Anfrage aus und es kann Rechenzeit eingespart werden. Weiterhin ist es möglich, statisch generierte Inhalte leichter auf verschiedene Server zu verteilen, als es mit Webanwendungen der Fall wäre, da das Anbieten von statischen Inhalten keine Installation von systemabhängiger Server-Side-Software benötigt. [46]

2.3.5 Client-Side-Rendering

Im Gegensatz zur Vorgehensweise von Static-Site-Generatoren wird im „Client-Side-Rendering“ meist nur ein Grundgerüst eines HTML-Dokumentes an den Anwender ausgeliefert [107]. Es enthält neben Informationen in der Kopfzeile meist nur ein sogenanntes „Anker-Element“, an das ein in JavaScript implementiertes Web-Framework anknüpft. Das Anker-Element stellt die Wurzel einer durch das Framework dynamisch generierten HTML-Struktur dar [107].

Der JavaScript-Quellcode, der beim Laden der Webseite durch den Nutzer mitgeliefert wird, ist in der Lage, die Haupt- und Unterseiten dynamisch auf dem Gerät des Client zu generieren. Dieses Vorgehen hat den Vorteil, dass bei der Navigation durch die Webseite keine weiteren Anfragen an den Server mehr geschickt werden müssen. Die zur Berechnung der Darstellungselemente benötigten Ressourcen werden im Client-Side-Rendering daher auf dem Endgerät des Anwenders beansprucht. [67]

Verzögerungen in der initialen Ladezeit durch große JavaScript-Pakete können durch Aufspalten und stufenweises Nachladen des Quellcodes verhindert werden [27].

2.4 Internetbrowser-Kompatibilität

Verschiedene Web-Browser unterstützen moderne Frontend-Technologien und -Standards in unterschiedlichem Ausmaß. Einen Überblick über die Unterstützung einzelner Funktionen innerhalb unterschiedlicher Browser und deren Versionen dieser lässt sich beispielsweise auf der Webseite „CanIUse“ finden [25].

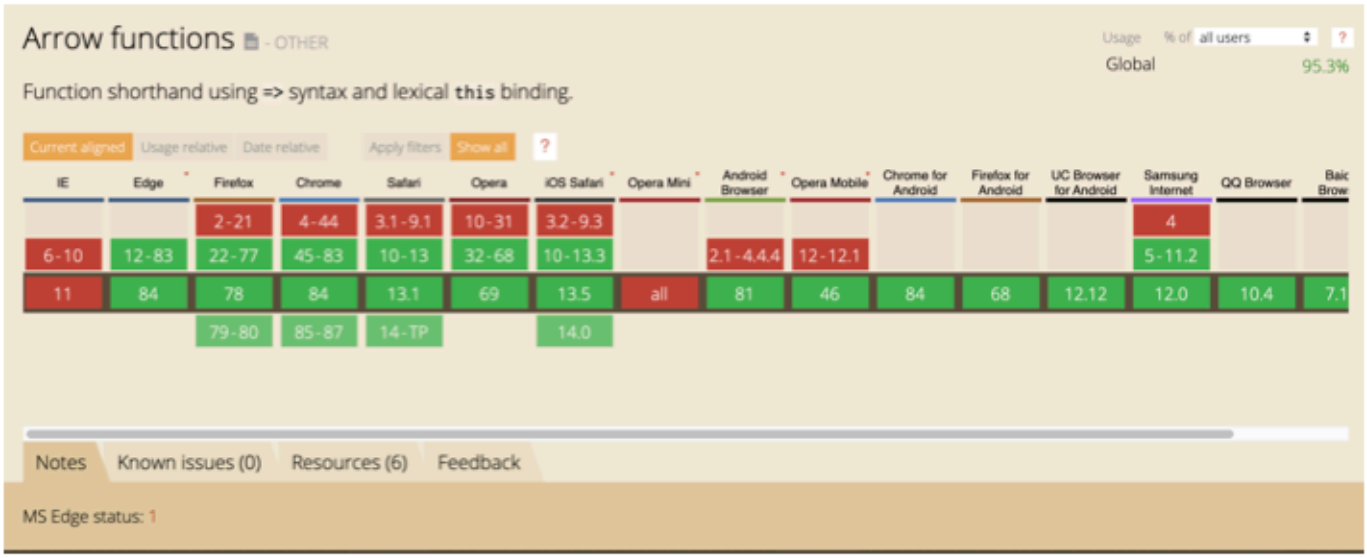

Abbildung 2.2: Statistik über die Unterstützung einer JavaScript-Funktionalität verschiedener Internetbrowser [11]

Googles „Chrome“-Browser besitzt sowohl weltweit den größten Marktanteil [10] als auch die umfassendste Konformität gegenüber Web-Spezifikationen [26]. Je nach Endgerät, Land und Region existieren allerdings abweichende Nutzungsstatistiken in Bezug auf sonstige Internet-Browser. So sind beispielsweise die Web-Browser „UC Browser“ und „QQ Browser“ in China mit einem Marktanteil von knapp 25 Prozent vertreten [9]. Diese Internetbrowser unterstützen beispielsweise nicht alle Funktionen, die im Chrome-Internetbrowser vertreten sind [23] [24]. Neben der Abdeckung von älteren Browsern und Browserversionen ist die Internetbrowser-Kompatibilität demnach unter anderem auch in der Internationalisierung und Lokalisierung von Bedeutung.

Um einem Webentwickler dennoch den Zugriff auf moderne JavaScript- und CSS-Funktionen zu ermöglichen, gibt es eine Reihe an Werkzeugen. Das Ziel, mit minimalem Mehraufwand möglichst viele verschiedene Browser abzudecken, realisiert man oft mit einem sogenannten Transcompiler oder „Source-to-Source-Translator“ [33].

Einen Überblick über diese Vorgehensweise liefert Martin Fowler in seinem Blog-Post „TransparentCompilation“. Laut Fowler ist ein „Source-to-Source“-Compiler oder -Übersetzer ein Werkzeug, welches den Quellcode einer Programmiersprache in den einer anderen überführt. Einige Compiler dieser Art ermöglichen es zusätzlich, nach Fowler, auf transparente Art und Weise Quellcode so zu übertragen, dass diesem nach der Übersetzung ein bestimmter Grad an Lesbarkeit und Nachvollziehbarkeit erhalten bleibt. Weiterhin führt Fowler aus, dass andere Compiler dieser Art durch das Entfernen von langen Variablen- und Funktionsnamen sowie von Leerzeichen eine kompaktere Version des Zielcodes erzeugen können. [33]

Ein weiter Ansatz ist die Generierung sogenannter „Source-Maps“ [64]. Im Artikel „Introduction to JavaScript Source Maps “ erläutert Ryan Seddon, Ein Frontend-Architekt der US-amerikanischen Firma „Zendesk“ eine Möglichkeit, eine Kombination aus komprimiertem und dennoch debug-fähigem Quellcode zu generieren [64]. Dafür erstellen unterstützende Compiler eine Datei, mit deren Hilfe man vom kompakten Quelltext auf

den ursprünglichen Quellcode schließen kann. Leitet man den Debug-Modus der entsprechenden Programmierumgebung ein, kann man so arbeiten, als wäre der Quelltext in seinem Originalzustand. [64]

2.4.1 Babel

Babel ist ein quelloffener JavaScript-Transcompiler. Dieser ursprünglich als „6to5“ bekannte Compiler wurde 2015 von Sebastian McKenzie veröffentlicht und ermöglicht die Umwandlung der damals jüngst veröffentlichten ECMAScript-2015-Spezifikation in die Version ECMAScript 5 (auch ES5) [102]. Zwischen Babel und dem seit 2016 de facto eingestellten Google-Produkt „Traceur“ konnte sich ersteres als Industriestandard etablieren [40]. Mit fast 18 Millionen Downloads pro Woche kommt es unter anderem bei Industriegiganten wie Facebook, PayPal und Netflix zum Einsatz [120] [91]. Babel erlaubt standardmäßig die Nutzung aller ECMAScript Funktionen bis einschließlich der Version ECMAScript 2020 [79]. Weiterhin ermöglicht Babel mithilfe einer Vielzahl von Plugins die Nutzung von experimenteller JavaScript-Syntax und ECMAScript-Proposals, welche sich noch in frühen Phasen der Vorgaben des TC39-Komitees befinden [75]. Über sogenannte „Transform Plugins“ werden dann Abschnitte eines Quelltextes automatisiert so umgeschrieben, dass sie zur ECMAScript-Version des Zielsystems kompatibel sind [75]. Sollte der Browser, auf dem der generierte Quelltext laufen soll, nicht in der Lage sein, ES5-kompatiblen JavaScript-Code auszuführen, können über sogenannte „Polyfills“ fehlende Funktionen nachgerüstet werden [12]. Das Wort Polyfill ist eine Wortneuschöpfung des „Introducing HTML5“-Co-Autoren Remy Sharp und beschreibt einen Code-Auszug, der eine Programmierschnittstelle einer Browserfunktion nachträglich bereitstellt, wenn diese nativ nicht vorhanden ist [116].

Babel in Verbindung mit Polyfills ermöglicht es also, JavaScript anhand neuester Standards und Spezifikationen unabhängig des Zielsystems zu schreiben.

2.4.2 PostCSS

Das folgende Unterkapitel ist sinngemäß der GitHub-Seite von „PostCSS“entnommen [94].

PostCSS ist ein Werkzeug, welches ein CSS-Stylesheet in einen abstrakten Syntaxbaum übersetzt und es auf JavaScript basierenden Plugins erlaubt, diesen zu analysieren und modifizieren. Diese Plugins können einen einheitlichen Programmierstil von CSS-Regeln gewährleisten, häufige Fehler melden oder den Einsatz von „Don’t repeat yourself“ (DRY)-konformen Paradigmen wie Variablen und Mixins selbst in solchen Browsern ermöglichen, die diese Funktionen nicht nativ unterstützen. Weiterhin ist es möglich, mit PostCSS experimentelle Funktionalitäten aus neuesten Arbeitsentwürfen von CSS-Spezifikationen zu benutzen. Syntax, die nach den CSS-Spezifikationen grundsätzlich unzulässig wäre, wird durch PostCSS in korrektes CSS übersetzt.

Die mittlerweile über 200 existierenden Erweiterungen bieten verschiedenste Möglichkeiten. Plugins wie „Autoprefixer“ und „postcss-reset“ eliminieren beispielsweise Unterschiede im Verhalten von CSS-Regeln und -Eigenschaften zwischen unterschiedlichen

Internet-Browsern [93] [56]. Andere, unter ihnen „RTLCSS“, welches die standardmäßige Lese- und Schreibrichtung von „links nach rechts“ umkehrt, erleichtern die Arbeit mit Inhalten in unterschiedlichen Sprachen [118].

2.5 React

Die folgenden Unterkapitel sind sinngemäß aus der Dokumentation von „React“ übernommen [103].

React ist eine quelloffene JavaScript-Bibliothek der amerikanischen Social-Media Platform „Facebook“. Durch sogenannte „Komponenten“ erlaubt React es, zustandsbehaftete Templates zu definieren und durch diese eine Webanwendung deklarativ aufzubauen. Das Verhalten einer Komponente bei Änderung ihres Zustandes wird dabei von React verwaltet und kann durch von React gelieferte Funktionen beeinflusst werden.

2.5.1 Grundlagen

React Komponenten können entweder über eine klassen- oder funktionsbasierte Schreibweise definiert werden. Ihre Darstellung im Browser wird durch eine XML-basierte Form von JavaScript beschrieben, die sich „JSX“ nennt. JSX liefert die Möglichkeit, einzelne Komponenten durch native HTML Elemente wie `li` und `div` als auch durch andere, bereits definierte Komponenten zu beschreiben. In der Ausführung des Programmcodes werden diese dann rekursiv durch die native HTML-Elemente ersetzt.

React erlaubt es weiterhin, Komponenten mit verschiedenen Zuständen zu versehen, um sie dynamisch oder interaktiv zu gestalten. Diese „State“ genannten Variablen sind zunächst unveränderlich und können nur mit von React bereitgestellten Methoden verändert werden. Ändert sich der Zustand einer Komponente durch einen solchen Methodenaufruf, so wird sie automatisch von React aktualisiert und mit dem neuen Wert des Zustandes dargestellt.

2.5.2 Komponenten

Zur Definition von Komponenten stellt React zwei verschiedene Möglichkeiten zur Verfügung.

Die erste klassenbasierte Variante wird durch die JavaScript-Klassensyntax beschrieben. Um eine neue Komponente anzulegen, muss diese von der abstrakten Klasse „React.Component“ erben. Es ist dann möglich, mit sogenannten „Life-Cycle-Hooks“ in den Lebenszyklus dieser Komponente einzugreifen und entsprechend auf verschiedene Aktivitäten zu reagieren. Über JSX in der „Render“-Methode kann dann die Darstellung der Komponente beschrieben werden.

Funktionsbasierte Komponenten werden durch JavaScript-Funktionen beschrieben, die JSX als Rückgabewert liefern. Seit Version 16.8 liefert React zudem die Möglichkeit, auch funktionale Komponenten zustandsbehaftet zu definieren. Zuvor konnten funktionsbasierte Komponenten nur zustandslos beschrieben werden [51].

2.5.3 Create-React-App

Das folgende Unterkapitel ist sinngemäß aus der Dokumentation von „Create-React-App“ entnommen [36].

„Create-React-App“(CRA) ist eine auf Node.js basierende Kommandozeilenanwendung des React-Entwicklerteams zur Initialisierung von React-Projekten [15]. CRA liefert dabei verschiedene Projektschablonen, in denen die zur Ausführung nötigen Abhängigkeiten und deren Konfigurationsdateien vordefiniert sind. Damit entfällt der Konfigurationsaufwand, der bei der Erstellung komplizierter Projekte nötig wäre. CRA liefert weiterhin vordefinierte Kommandozeilenparameter zur Ausführung und Kompilierung des erstellen Projektes über das Kommandozeilenwerkzeug „npm“. Um die im Projekt genutzten JavaScript-Module, Resourcen und Abhängigkeiten zu einem Kompilat zu bündeln, verwendet CRA den JavaScript-Modul-Packer „Webpack“ [113]. Webpack ist weiterhin in der Lage, durch Plugins JavaScript-Quelltext und Ressourcen wie Bildmaterial anzupassen, bevor sie in das Bündel eingefügt werden. Ein solches Plugin wird „Loader“ genannt [61]. So verwendet ein durch CRA erstelltes Projekt beispielsweise Webpack-Loader, die über Babel (siehe 2.4.1) JavaScript-Quelltext in die ECMAScript-Version ES5 übersetzen. Weiterhin wird innerhalb eines CRA-Projektes PostCSS (siehe 2.4.2) genutzt, um eine übereinstimmende Funktion von CSS-Regeln für verschiedene Internetbrowser zu gewährleisten [76].

2.6 Content-Management-Systeme

Laut der Definition von Margaret Rouse, einer Mitarbeiterin der amerikanischen Marketing-Firma „TechTarget“, ist ein „Content-Management-System“ (CMS) eine Anwendung, die es ermöglicht, Inhalte wie Text-, Bild- und Videomaterial innerhalb einer Webseite oder -anwendung zu ändern, zu erstellen oder zu verwalten [81]. Nach ihr ist ein CMS auf dem Client-Server-Model aufgebaut. Sein Frontend, die sogenannte „Content-Management-Application“ (CMA), ist eine Benutzeroberfläche, welche es erlaubt, bestimmte Bereiche einer Webseite und deren Inhalte anzupassen, ohne dass eine Änderung am Quelltext notwenig ist. Die sogenannte „Content-Delivery-Application“ (CDA) verwaltet laut Definition die von der CMA gesendeten Daten und speichert diese und andere benötigte Daten in einer Datenbank ab. Weiterhin verwaltet sie die Zusammensetzung und die strukturierte Auslieferung der Daten. [81]

2.7 Scrivito CMS

Das folgende Unterkapitel ist, wenn nicht anders markiert, an das von Scrivito veröffentlichte White-Paper „JAMStack for Web Projects“ angelehnt [89].

Scrivito ist ein sogenanntes „cloud-basiertes“ Content-Management-System. Das bedeutet, dass sein Backend, also die CDA, nicht auf einem eigenen Web-Server gehostet werden muss. Stattdessen liegt es auf einem durch Scrivito verwalteten Cloud-Server. [86]

Scrivito ist auf einem Software-Stack (siehe 2.3.1) aufgebaut, der mit dem Namen „JAM-Stack“ beziffert wird. Dieser besteht aus den Softwarekomponenten **J**avaScript, **A**PI („Application-Programming-Interface“) und HTML-„**M**arkup“.

HTML-Markup Nach dem Konzept des JAMStack wird bei dem Aufruf einer Scrivito-Webseite deren HTML-Struktur – anders als bei dynamischen Webseiten (siehe 2.3.2) – nicht mehr pro Webseitenaufruf neu erzeugt. Stattdessen werden Inhalte, ähnlich zu einem Static-Site-Generator (vergleiche 2.3.4) vorgeneriert. Wird eine Änderung an Daten vorgenommen, die im Content-Management-System gespeichert sind, werden die entsprechenden Seiten und deren Inhalte automatisch neu von Scrivito erzeugt.

Die so generierten Inhalte werden dann über ein sogenanntes „Content-Delivery-Network “ (CDN) bereitgestellt. Ein CDN ist ein globales Netzwerk von Servern, welche Inhalte an einem Standort anbieten, welcher dem Nutzer geografisch am nächsten ist. Weiterhin wird durch eine hohe Anzahl von CDN-Servern gewährleistet, dass die verwalteten Inhalte auch bei Ausfall eines Servers im CDN verfügbar sind. [114] Damit kann die Ladezeit und Verfügbarkeit einer JAMStack-basierten Website verbessert werden.

API Zur Programmierung von Webseiten liefert Scrivito die sogenannte „Scrivito-SDK“ („Software-Development-Kit “). Diese stellt alle in Scrivito gespeicherten Inhalte zur Verfügung [1]. Sie liefert weiterhin die Benutzeroberfläche (auch CMA) des Content-Management-Systems. Diese funktioniert unter anderem über einen sogenannten „What-You-See-Is-What-You-Get“ (WYSIWYG)-Editor, über den die Webseite aufgebaut werden kann [8]. Ein WYSIWYG-Editor ist in der Lage, schon während der Erstellung der Webseite dem Editor eine Ansicht zu bieten, die der finalen Darstellung ähnelt [53]. Bei der Anpassung von Inhalten in der CMA werden entsprechende Informationen über die Scrivito-SDK an das Backend übertragen und dort gespeichert [1]. Weiterhin liefert sie einen programmatischen Zugriff auf Scrivitos Inhalte über JavaScript [87].

JavaScript In Scrivito werden Haupt- und Unterseiten durch sogenannte „Objects“ (im weiteren Verlauf auch „Objekte“) dargestellt. Der Inhalt von Scrivito-Objekten wird durch sogenannte „Widgets“ aufgebaut. Widgets und Objekte verwalten verschiedene Inhalte, Funktionalitäten und Konfigurationsmöglichkeiten. [88]

Zur Implementierung von Widgets und Objekten wird React genutzt. React erlaubt es unter anderem, Widgets und Objekte komponentenweise aufzubauen [88]. Weiterhin ist es möglich, mit React statische Inhalte, die von Scrivito erzeugt wurden, nachträglich auf dem Gerät des Nutzers mit neuen Daten zu befüllen [90]. Diesen Prozess nennt man „Hydrierung“ [77]. Damit ist Scrivito in der Lage, den Großteil einer Webseite statisch an den Nutzer zu liefern und nur JavaScript für solche Teile einzubinden, welche auf der Client-Side gerendert werden müssen [90]. Scrivito vereint somit verschiedene Vorteile von statischen und dynamischen Webseiten.

2.8 Internationalisierung

„In a nutshell, localization revolves around combining language and technology to produce a product that can cross cultural and language barriers. No more, no less." – Bert Esselink [31]

Das folgende Unterkapitel orientiert sich inhaltlich am Artikel „Understanding Internationalization" von Michael Satran, einem Mitverfasser der technischen Dokumentation von Microsoft [85].

„Internationalisierung" (auch: *i18n* [101]) ist die Anpassung von Softwareprogrammen an die Anforderungen verschiedener Gruppen von Standorten und dazugehöriger Sprachen. Diese Anforderungen beinhalten unter anderem die Übersetzung von sprachlichen Inhalten oder die Umrechnung von Währungen oder Zeit- beziehungsweise Kalenderangaben. Weiterhin können sie die Anpassung von Bild-, Ton- und Videomaterial oder auch Wortspielen an die örtliche Kultur enthalten [99]. Eine Zusammenfassung dieser Anforderungen wird „Locale" genannt. Eine Locale kann über verschiedene Kennungen unterschieden werden [21].

Ein internationales Softwareprogramm besteht aus einem „globalen Programmkern", der unabhängig von sprachenabhängigen Inhalten die Funktionsweise des Softwareprogrammes implementiert. Der Prozess der Entwicklung dieses globalen Programmkerns wird als „Globalization" bezeichnet. Die „Lokalisierbarkeit" bezeichnet dabei das Maß dafür, inwiefern die Inhalte von Elementen – beispielsweise Zeichenfolgen oder Benutzeroberflächen des Softwareprogrammes – in andere Sprachen übersetzt werden können. Wichtig ist dabei nicht nur die Möglichkeit der Anpassung ohne eine Änderung am Quelltext, sondern auch die Stabilität und Integrität der umgebenden Elemente, beispielsweise beim Umbruchverhalten langer Zeichenketten (siehe Abbildung 2.3). Kann ein korrektes Umbruchverhalten nicht gewährleistet werden, so handelt es sich um einen sogenannten „Layout-Fehler" [4, Kap. 2]. Der konkrete Vorgang, bei dem Inhalte übersetzt und angepasst werden, wird „Lokalisierung" genannt. Ähnlich zum Vorgang der Internationalisierung besitzt die Lokalisierung die Abkürzung *L10n* [101].

Abbildung 2.3: Ein Layout-Fehler bei der Lokalisierung [4, Kap. 2]. Links: britische Version. Rechts: mexikanische Version

3 Analyse

3.1 [redacted]-Landing-Page

3.1.1 Ist-Analyse

Da es sich bei dem Joint-Venture [redacted] um ein relativ junges Unternehmen handelt, besitzt es noch keine sogenannte „Landing-Page“, um von potenziellen Interessenten im Internet gefunden werden zu können. Bei einer Landing-Page handelt es sich meist um eine Internetseite, die zu Werbezwecken ein Unternehmen vorstellt, Informationen liefert, dessen Dienstleistungen und Produkte aufzeigt und Kontaktmöglichkeiten bietet [115]. Sie ist zum Zeitpunkt dieser Ausarbeitung noch nicht vorhanden und soll erarbeitet werden. Zur Umsetzung der Webseite wurde von Ergosign unternehmensintern ein Design entwickelt[1]. Texte, Video- und Bildmaterial sind zum Zeitpunkt dieser Ausarbeitung noch nicht bekannt. Sie werden nachträglich bereitgestellt.

3.1.2 Soll-Analyse

Damit das oben genannte Text, Video- und Bildmaterial von Nicht-Programmierern eingebunden und verändert werden kann, muss die [redacted]-Landing-Page über ein Content-Management-System verfügen. Da sich die Ergosign GmbH die Anforderung einer „pixelperfekten“ Umsetzung von Designvorgaben stellt, muss das gewählte Content-Management-System in der Lage sein, die Vorgaben einwandfrei umsetzen zu können. Weiterhin handelt es sich bei den Unternehmenspartnern Artop und Ergosign um je ein chinesisches und ein deutsches Unternehmen. Das Unternehmen [redacted] existiert deshalb im internationalen Raum. Aufgrund dessen muss die Landing-Page mehrsprachig verfügbar sein. Eine besondere Anforderung an die Webseite ist darüber hinaus, dass sie in den Webbrowsern „Google Chrome“, „Firefox“ und „Safari“ sowie deren mobilen Versionen funktionieren soll. Zusätzlich soll die Funktionalität der Webseite in den in China genutzten Desktop- und Mobile-Browsern „QQ“ und „Baidu“ so gut wie möglich gewährleistet werden. Zudem wird jeweils die Umsetzung einer Ansicht für einen Desktopcomputer, ein Tablet und ein mobiles Endgerät erarbeitet. Die Entwicklung einer Webseite für verschiedene Größen von Endgeräten wird genannt „Responsive Design“ [78]

Im Anschluss soll für die Ergosign GmbH eine kurze Evaluierung von Scrivito erarbeitet werden. In dieser werden eventuelle Schwachstellen von Scrivito und Schwierigkeiten

[1] Die Datei der Designvorgaben kann unter der Adresse https://gitlab.ergosign-projects.com/ergosign/jk-bachelorthesis/-/tree/master/Assets gefunden werden

bei der Umsetzung der [redacted]-Landing-Page mit dem Content-Management-System dargestellt.

3.2 Internationalisierung

3.2.1 Ist-Analyse

Eine initial eingerichtete Scrivito-Instanz bietet keine Möglichtkeit, Haupt- und Unterseiten mit verschiedensprachigen Inhalten zu verwalten. Es existiert allerdings ein vom Scrivito-Team bereitgestellter Leitfaden zum Kopieren von bereits vorhandenen Seiten an eine URL mit sprachenbasiertem Präfix [16]. Der Inhalt des folgenden beiden Unterabschnitte ist diesem Leitfaden entnommen.

Verschiedensprachige Inhalte werden laut diesem Leitfaden durch Namensräume unterschieden und in Scrivitos „Hierarchy-Browser“ dargestellt [105]. Mehrsprachige Haupt- und Unterseiten sind innerhalb eines Wurzelobjektes gegliedert und können durch ihren Pfad unterschieden werden. Das Wurzelobjekt besitzt den Pfad „/“ und wird von Scrivito, wenn nicht anders konfiguriert, initial beim Aufruf dargestellt [19]. Um sprachenabhängige und -unabhängige Inhalte zu trennen, werden erstere mit identifizierenden Sprachkürzeln wie „de“ und „en“ und einem Präfix versehen. Eine deutsche Hauptseite besitzt so etwa den Pfad „/lang/de“. Der Leitfaden beschreibt weiterhin die Implementierung eines Navigationselementes zur Steuerung der angezeigten Seite. Ebenfalls dargestellt ist ein Beispiel einer Funktion zur dynamischen Bestimmung der initial dargestellten Hauptseite anhand der im Browser eingestellten, bevorzugten Sprache eines Nutzers. Überdies wird eine Funktion beschrieben, um anhand von Subdomains wie zum Beispiel „de.domain.com“ die passende Seite zu ermitteln.

Zur Umsetzung der in dem Leitfaden vorgeschlagenen Methoden gibt es keine grafische Benutzeroberfläche. Alle dargestellten Interaktionen müssen entweder in der Entwicklerkonsole des Browsers ausgeführt oder im Quelltext angepasst werden. Es existieren keine vorgegebenen Konfigurationsmöglichkeiten. Weiterhin ist nicht beschrieben, wie mit Fehlern umzugehen ist. Es gibt ebenfalls keine Möglichkeit, sprachenabhängige Haupt- und Unterseiten an ein anderes Sprachpräfix zu verschieben, anstatt zu kopieren. Ebenso ist es nicht möglich, die sprachenunabhängigen Haupt- und Unterseiten zu verwalten oder eine Übersicht über diese zu gewinnen. Diese Punkte erschweren die Benutzung von der von Scrivito bereitgestellten Internationalisierungsfunktionen für einen Endnutzer.

3.2.2 Soll-Analyse

Must-Have

Editoren und Redakteure sollen in der Lage sein, durch ein „Internationalisierungsplugin“ mehrsprachige Versionen von bereits existierenden Haupt- und Unterseiten über

eine grafische Benutzeroberfläche zu erstellen. Es soll mit minimalem Programmier- und Konfigurationsaufwand bedient werden können. Zudem soll es möglich sein, zusätzlich zu Scrivitos eingebautem Hierachy-Browser eine Übersicht aller Haupt- und Unterseiten einschließlich ihrer Pfade und Permanentlinks zu erhalten. In dieser Übersicht soll es dem Benutzer darüber hinaus möglich sein, Haupt- und Unterseiten innerhalb eines baumartig aufgebauten Konfigurationsmenüs auszuwählen und über weitere Menüpunkte an einen Pfad zu kopieren, der eine noch nicht vorhandene Sprache eindeutig identifiziert. Die Auswahl einer Haupt- oder Unterseite soll alle in der Baumhierarchie untergeordneten Zweige einschließen. Wählt der Nutzer also eine Seite aus, die andere Seiten beinhaltet, sollen diese beinhalteten Unterseiten ebenfalls automatisch mit ausgewählt werden.

Programmierer sollen in der Lage sein, jedes Widget bei seiner Definition mit dem Plugin kompatibel zu machen. Dazu sollen durch das Plugin spezielle Java-Script-Funktionen bereitgestellt werden. Über das Konfigurationsmenü eines so entstandenen Widgets sollen Informationen einsehbar sein, die durch das Plugin verwaltet werden. Das Plugin soll ebenfalls in der Lage sein, die bei der Kopie von Haupt- und Unterseiten ebenfalls kopierten, beinhalteten Widgets miteinander zu verknüpfen. Beinhaltet eine Seite beispielsweise ein Widget „X“ und wird durch die Erweiterung an einen neuen Pfad kopiert, so sollen „X“ und seine Kopie mit einer ihnen innerliegenden ID referenziert werden. Damit kann auf ein kopiertes Widget auf seinen Ursprung zurück geschlossen werden.

Could-Have

Neben den grundlegenden Funktionen die dieses Plugin bereitstellen soll, existieren noch weitere Anforderungen, welche jedoch nicht zwingend für den Einsatz des Plugins am Ende der Thesis fertiggestellt sein müssen. Das Plugin könnte ebenfalls die Möglichkeit bieten, über ein Konfigurationsmenü alle Texte der Webseite, die sich innerhalb von Widgets befinden, welche mit dem Plugin kompatibel sind, in ein spezielles Dateiformat zu exportieren. Damit könnte es möglich sein, Texte nicht nur direkt auf der Webseite, sondern extern durch spezielle Übersetzungswerkzeuge zu übersetzen. Nach dem Übersetzungsvorgang ließen sich die Texte wiederum in die Seite einzugliedern. Weiterhin könnte das Plugin in der Lage sein, innerhalb eines Konfigurationsmenüs die Sprache zu konfigurieren, deren Seite initial beim Aufruf der Webseite dargestellt wird. Alternativ könnte man auf die Option ausweichen, die in 3.2.1 beschriebene Funktion zur automatischen Ermittlung der Startseite nach der bevorzugten Sprache des Nutzers weiter zu konfigurieren.

4 Stand der Technik

Dieses Kapitel stellt die Internationalisierungsfunktionen des Content-Mangagement-Systems „Wordpress“ dar. Die dazu nötigen Informationen wurden aus der Bedienungsanleitung für „GNU gettext“ und dem Anleitenden Artikel „I18n for Wordpress Developers“ entzogen [38] [47].

Die Internationalisierung eines Wordpress-Systemes geschieht über das Computerprogramm „gettext“. Es ist in der Lage, Zeichenketten aus Quelltext zu filtern, wenn diese in speziellen gettext Funktionen eingesetzt werden. Nach dem Filterprozess werden die Zeichenketten innerhalb sogenannter „Nachrichtenkataloge“ gespeichert. Eine Wordpress-Instanz verwaltet diese Nachrichtenkataloge im Format „Portable-Objects-Template“ (POT). Eine POT-Datei stellt einen Nachrichtenkatalog dar, in dem sich nur eine Grundstruktur und noch keine Übersetzungen befinden. POT-Dateien können entweder unmittelbar innerhalb eines Texteditors oder über speziell für sie angefertigte Software zum Übersetzungsprozess genutzt werden. Aus diesem Vorgang wird eine „Portable-Object“ (PO)-Datei erzeugt. Diese enthält im Gegensatz zur POT-Datei übersetzte Zeichenketten. Wordpress kann diese PO-Dateien nutzen, um internationale Inhalte an den entsprechenden Stellen einzusetzen. Um die Laufzeit des Zugriffes auf PO-Dateien zu verbessern, können in binäre „Machine-Object“ (MO)-Dateien übersetzt werden. Ein Beispiel für eine PO-Datei befindet sich in der unteren Abbildung.

Listing 4.1: Beispiel für eine PO-Datei

```
"Hier stehen Informationen der Kopfzeile"
"Tanslator: Julian Krieger"
"Language: de"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: src/beispiel.php"
# Übersetzer-Kommentar: Eine Frage zum Text?
msgid "This string can be translated."
msgstr "Dieser Text ist in der Lage, uebersetzt zu werden."

#: src/wp_button_widget.php"
msgid "Click me!"
msgstr "Drück mich!"
```

Kommentare, die mit " beginnen und mit " enden, bezeichnen Informationen der Kopfzeile. Alle anderen Kommentare werden mit dem Rautezeichen „#" geschrieben. Befindet sich ein „:" hinter dem Rautezeichen, so bezieht sich der Kommentar auf den Quelltext. Ein Leerzeichen hinter dem Rautezeichen bezeichnet einen Übersetzer Kommentar. Textstellen werden durch eine *msgid* gekennzeichnet. Der übersetzte Text befindet sich unmittelbar hinter der *msgid* und wird mit dem Schlüsselwort *msgstr* identifiziert.

Der typische Vorgang der Übersetzung über PO-Dateien wird durch Leiva und Alabau im Paper „Automatic Internationalization for Just In Time Localization of Web-Based User Interfaces" dargestellt [58]. Die Informationen des folgenden Abschnittes sind semantisch daraus entnommen.

Abbildung 4.1: Übersetzungsvorgang über PO-Dateien [58]Braun: Entwickler – Gelb: Übersetzer – Blau:Endnutzer

Er zeigt die Entnahme der Textinhalte (1), ihre anschließende Übersetzung durch einen Übersetzer (2) und die Wiedereinbindung der Inhalte in die Benutzeroberfläche oder Webseite (3). Am Ende des Vorgangs befinden sich Qualitätssicherungsprozesse und Testverfahren durch den Entwickler (4) und den Übersetzer (5).Dabei werden die Schritte 3 bis 5 so lange wiederholt, bis eine ausreichende Qualität gewährleistet ist. Anschließend können die Textinhalte wieder in die Seite eingebettet (6) und durch Endnutzer getestet werden (7). Da die Übersetzung und Qualitätssicherung meist von externen Übersetzern übernommen wird, hat dieser Vorgang bei einer hohen Anzahl an Iterationen den Nachteil, dass hohe Kosten und Zeitverzögerungen entstehen können. Ein solcher Übersetzungsprozess hat weiterhin den Nachteil, dass er entkoppelt von der Darstellung der Webseite durchgeführt wird. Entsteht während des Übsersetzungsprozesses ein Fehler, kann er meist erst zu einem späteren Zeitraum entdeckt werden.

5 Konzeption

5.1 Internetpräsenz des Joint Ventures „[redacted]“

5.1.1 Einleitung

Die im folgenden Unterkapitel benutzten Begriffe und Methoden stammen aus dem Buch „Atomic Design“ von Brad Frost [35]. Frost definiert den Aufbau eines Designsystems als eine Zusammensetzung verschiedener, wiederverwendbarer Elemente. Diese gliedert er aufsteigend nach Komplexität, Funktionsweise und Aufbau. Alle Unterarten von Elementen können unter dem Oberbegriff „Komponente“ zusammengefasst werden.

Frost bezeichnet eine Komponente, die als Grundbaustein dient und nicht in kleinere Teile zerlegt werden kann als „Atom“. Ein Atom besitzt eine Reihe von Attributen und liefert genau eine Funktionsweise. Es wird benutzt, um alle anderen Komponenten aufzubauen. Eine Gruppe, welche die Funktionen der enthaltenen Atome zu einer neuen Funktion kombiniert, nennt er „Molekül“. Frost definiert die nächste Stufe der komplexen Abstraktion als „Organismus“. Er definiert einen Organismus als eine „relativ komplexe Komponente der Benutzeroberfläche, welche Atome, Moleküle und auch andere Organismen in sich vereint “. Im Gegensatz zu Molekülen und Atomen formt ein Organismus einen konkreten, eigenständigen Abschnitt der Benutzeroberfläche. Als Beispiel nennt er die Navigations- oder Suchleiste, die auf vielen Webseiten vorhanden ist. Ein Objekt, welches Struktur, Anzahl und Anordnung der Organismen, Moleküle und Atome zueinander, nicht aber deren Inhalt definiert, abstrahiert Frost als „Schablone“. Sie kann als Grundgerüst verstanden werden. Füllt man die Schablone mit Inhalt, so Frost, entsteht eine sogenannte „Seite“. Eine Seite zeigt die endgültige Darstellung der Benutzeroberfläche mit konkreten Inhalten wie Bildern, Text und Interaktionen an.

Um einen einheitlichen Wortschatz zur Strukturierung von Komponenten zu gewährleisten, nutzt Ergosign feststehende Begriffe für bestimmte Arten von Komponenten. Sie lehnen an „Atomic Design“ an und fassen Atome als „Kontrollelemente“ („Controls“), Moleküle und Organismen als „Bausteine“ („Building-Block“) und Schablonen und Seiten als „Ansichten“ („View“) zusammen.

5.1.2 Umsetzung

Zur Entwicklung der Webseite liegt eine Datei vor, die den detaillierten Aufbau der zu entwickelnden Internetpräsenz dokumentiert. Aus ihr können alle nötigen Informationen wie beispielsweise Schriftarten, Farben und Bilder abgeleitet werden. Zusätzlich zeigen die Designvorgaben die Darstellung aller Elemente auf einem Desktop-Computer mit der

Auflösung „1920 × 1080 Pixel“ und einem mobilen Endgerät mit der Auflösung „360 × 640 Pixel“. Um die Spanne zwischen diesen Auflösungen geringer zu halten, reagieren einige Elemente zusätzlich auf die Auflösungsanforderung „768 × 1024 Pixel“, was der der Bildschirmgröße der meisten Tablets entspricht [98]. Die Verhaltensweisen der Elemente, die auf die letztere Auflösung reagieren, ist *nicht* in den Designvorgaben enthalten und wurde im Rahmen dieser Bachelorthesis erarbeitet.

Um die Implementierung modular gestalten und schrittweise umsetzen zu können, werden die existierenden Designvorgaben zunächst in atomare Elemente eingeteilt. Der Inhalt eines atomaren Elementes wird später zur Laufzeit durch den Editor oder Redakteur spezifiziert. Um die Designvorgaben nahtlos umzusetzen, werden folgende Kontrollelemente benötigt:

Ein Textkontrollelement zur Darstellung von Textinhalt. Um den Designvorgaben gerecht zu werden, bietet es Konfigurationsmöglichkeiten zur Auswahl von Textfarbe, Ausrichtung, Schriftgröße, Schriftstil und -art, sowie Zeilenhöhe und Zeichenabstand. Die Designvorgaben sehen zudem die Möglichkeit vor, Text um 90 oder 270 Grad drehen zu können. Weiterhin kann ein Textkontrollelement neben lateinischen Buchstaben auch Zeichen des chinesischen *Han*-Systems enthalten. Im chinesischen Schriftsystem kann von links nach rechts, aber auch Vertikal geschrieben werden. Deshalb bietet jedes Kontrollelement dieser Art zusätzlich die Möglichkeit, die Schreibrichtung und die Textausrichtung zu konfigurieren.

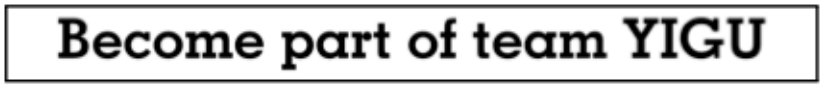

Abbildung 5.1: Ein Textkontrollelement

Ein Bildkontrollelement zur dynamischen Einbettung einer Grafik. Das Bild und seine Größe sind zum Kompilierungszeitpunkt des Programmes nicht bekannt. Es wird zur Laufzeit durch einen Nutzer eingefügt. Der Benutzer soll in der Lage sein, die Größe des Bildkontrollelements entweder auf die tatsächliche Größe seines Bildes oder auf die Größe des umgebenden Container-Elements festzulegen.

Abbildung 5.2: Ein Bildkontrollelement

Ein Darstellungskontrollelement zur kontrollierten Darstellung von enthaltenen Komponenten bei verschiedenen, festen Bildschirmgrößen. Dieses Kontrollelement ermöglicht

es, enthaltene Komponenten in Abhängigkeit von der Breite des verwendeten Internetbrowser-Fensters ein- und auszublenden. Um die Arbeit des Editors oder Redakteurs einfacher zu gestalten, verfügt es zusätzlich über die Funktion, die enthaltenen Komponenten im Editor-Modus selbst dann anzuzeigen, wenn sie laut der aktuellen Gerätekonfiguration nicht dargestellt werden würden.

Ein Kontrollelement mit zweispaltigem Inhalt Dieses Kontrollelement liefert die Möglichkeit, die darin enthaltenen Komponenten zweispaltig darzustellen und zu verwalten. Dabei sollen der Innenabstand der Spalten zueinander, ihre Mindestgröße und ihr Umbruchverhalten konfigurierbar sein.

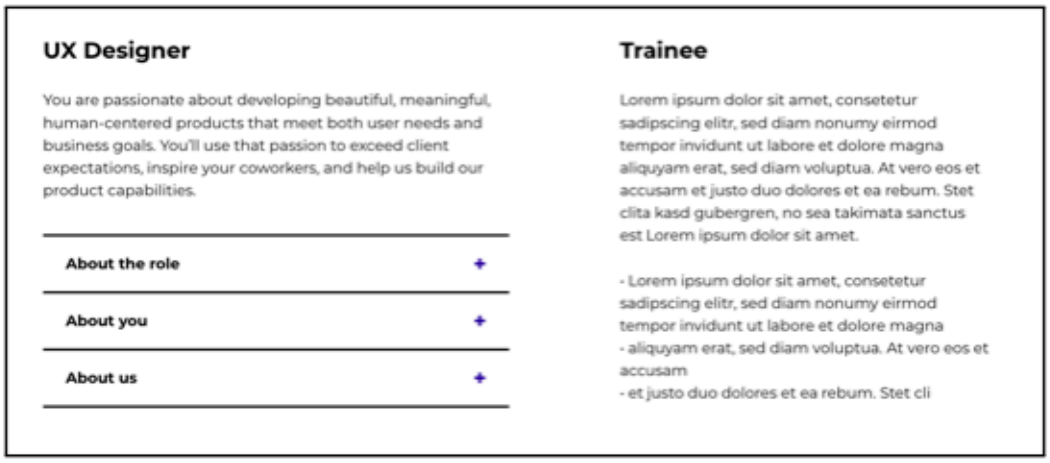

Abbildung 5.3: Ein Spaltenkontrollelement mit Inhalt

Ein Kontrollelement zur konfigurierbaren Anordnung von darin enthaltenen Komponenten innerhalb einer Reihe. Diese Komponenten sollen hierbei entweder horizontal oder vertikal angeordnet angezeigt werden. Ihre Ausrichtung auf der horizontalen beziehungsweise vertikalen Achse soll dabei konfigurierbar sein. Außerdem soll es möglich sein, des Kontrollelementes festzulegen, falls nicht ausreichend Platz in der ausgewählten Richtung verfügbar ist, um alle enthaltenen Komponenten darzustellen.

Abbildung 5.4: Ein horizontales Reihenkontrollelement mit Inhalt

Ein Abstandskontrollelement zur Einstellung von Außen- und Innenabstand der enthaltenen Komponenten. Dieses Kontrollelement bietet Einstellungsmöglichkeiten zur Konfiguration seines Außenabstandes zu umliegenden Komponenten. Weiterhin ist es möglich, einen Innenabstand dieses umfassenden Kontrollelementes zu seinen enthaltenen Komponenten festzulegen. Der Innenabstand bezieht sich dabei auf den durch dieses Element erzeugten Rahmen zu seinen Kinderkomponenten.

Aus den so definierten Kontrollelementen können alle benötigten Bausteine zusammengesetzt werden. Somit ist es beispielsweise möglich, aus zwei Textkontrollelementen einen Baustein zu erzeugen, der einen Titel und einen dazugehörigen Text verwaltet. Bausteine dieses Konzeptes, die Kontrollelemente und andere Bausteine in sich vereinen, beeinflussen meist ausschließlich deren Anordnung. Zur Umsetzung der Designvorgaben werden folgende Bausteine benötigt:

Ein Icon-Baustein Dieses Element besteht aus einem Icon, einem optionalen Titel-Textelement und einem optionalen Textelement, welches zusätzliche Informationen liefert. Icon, Titel und Text sind aufeinander abgestimmt und werden automatisch zentriert zueinander ausgerichtet. Sind Titel und Text nicht angegeben, sollen sie nicht dargestellt werden.

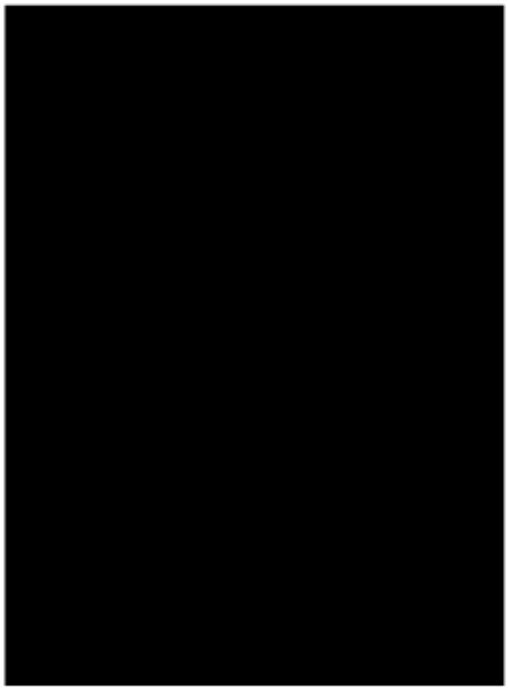

Abbildung 5.5: Ein Icon-Baustein mit Titel und Untertitel

Ein Icon-Zeile Baustein Zusätzlich zu einer Reihe von beliebigen Icon-Elementen beinhaltet die Icon-Zeile ein optionales Bild-Element und/oder ein Text-Element, welche als Titel der Icon-Zeile dienen. Hauptaufgabe der Icon Zeile soll es sein, die Icon-Elemente korrekt anzuordnen und ein Umbruchverhalten festzulegen.

Abbildung 5.6: Ein Icon-Zeile Baustein mit Titelbild und Untertitel

Ein aufklappbarer Textbaustein mit Zusammenfassung Dieser Baustein bietet die Möglichkeit, seinen Textinhalt in Folge einer Zusammenfassung des Inhaltes darzustellen. Zusätzlich ist es möglich, zu konfigurieren, ob er den Text initial im aufgeklappten Modus anzeigen soll oder nicht. Dieses Verhalten kann durch Konfigurationseigenschaften in Abhängigkeit der Bildschirmbreite verändert werden. Über das Interaktionselement „+„ an der rechten Außenseite kann der eingeklappte Modus zu „Aufgeklappt“ verändert werden. Danach wechselt das Interaktionselement zu „-„. Der Moduswechsel ist auch in die umgekehrte Richtung möglich.

Abbildung 5.8: Ein aufklappbarer Textbaustein im aufgeklappten Zustand

Ein scrollbarer Textbaustein dessen Inhalt auf kleineren Bildschirmgrößen aus dem Fenster herausragt. Benutzt der Nutzer sein Mausrad, kann er die Inhalte von links nach rechts scrollen lassen. Weiterhin ist es möglich, die Mindestgröße der Elemente und ihren Seitenabstand zueinander einzustellen. Um darzustellen, dass eine scrollbasierte In-

teraktion möglich ist, soll ein Teil des nicht vollständig gezeigten Inhaltes am rechten Bildschirmrand sichtbar bleiben.

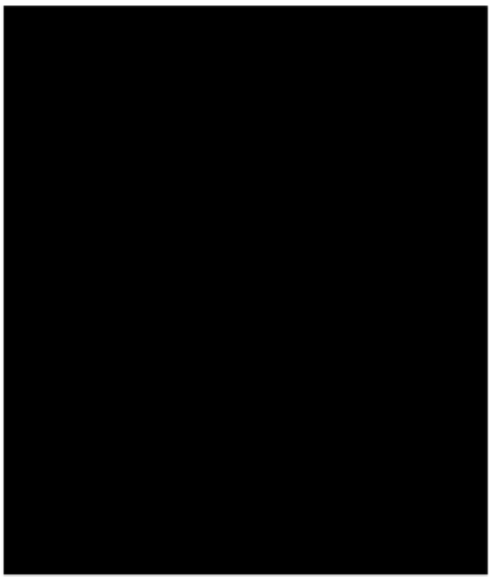

Abbildung 5.9: Ein scrollarer Baustein mit Inhalt

Ein Mosaikbildbaustein Dieser Baustein wird in den Designvorgaben nur einmal benötigt. Er definiert vier Positionen für Bilder sowie für Textelemente, die umlaufend um die Bilder positioniert sind. Diese können verschiedene Orientierungen annehmen.

Abbildung 5.10: Ein Mosaic-Bilderbaustein mit Inhalt

Eine Ansicht-Komponente Diese Komponenten werden benutzt, um alle Sektionen der Designvorgaben wiederzugeben. In den Designvorgaben werden sie durch einen vertikalen, orangenen Trennstrich gekennzeichnet und voneinander abgegrenzt. Daraus entstehen Abschnitte, die semantisch in die Kategorien „Startseite“, „Unternehmensvorstellung“, „Dienstleistungen“, „Schulungen“, „Portfolio“, „Ressourcen“, „Unternehmenskultur“, „Karriere“ und „Kontakt“ eingeteilt werden. Ein solcher Abschnitt beinhaltet in den meisten Fällen eine Überschrift, und Unterschrift und einen optionalen, einleitenden Text. Eine Ansicht definiert über ihre enthaltenen Bausteine und Kontrollelemente nur deren Position.

Abbildung 5.11: Der Abschnitt „Ressourcen“ mit Titel, Untertitel und einer Scroll-Reihe mit Inhalt

Abbildung 5.12: Der Abschnitt „Startseite“ mit Navigationsleiste, Logo, Titel, Untertitel und einem Knopfkontrollelement

5.2 Internationalisierungsfunktionen

Ziel der Ausarbeitung ist es, die Funktionen des in 3.2.1 vorgestellten Leitfadens zu erweitern und mittels eines Internationalisierungsplugins für Scrivito in einer grafischen Oberfläche zur Verfügung zu stellen. Weiterhin muss eine Werkzeugkette erstellt werden, um die gleichartige Funktionalität der Webseite in allen in 3.1.2 erwähnten Internet-Browsern zu so gut wie möglich realisieren. Da die Werkzeugkette implementierungsspezifisch ist, wird sie erst in Kapitel 6 weiter betrachtet.

Zur Benutzeroberfläche des Internationalisierungsplugins muss ein Modell erstellt werden, welches alle vorhandenen Funktionen darstellt. Zu diesen zählt eine Auswahl der Seite(n), die an ein neues Sprachpräfix verschoben oder kopiert werden sollen, sowie ein Auswahlelement des entsprechenden Präfixes. Weiterhin werden Funktionsknöpfe benötigt, über die entweder der Kopier- oder Verschiebevorgang eingeleitet werden kann. Schlussendlich muss ein Fenster implementiert werden, welches Meldungen anzeigt, die in den Vorgängen erzeugt werden (siehe Abbildung 5.13).

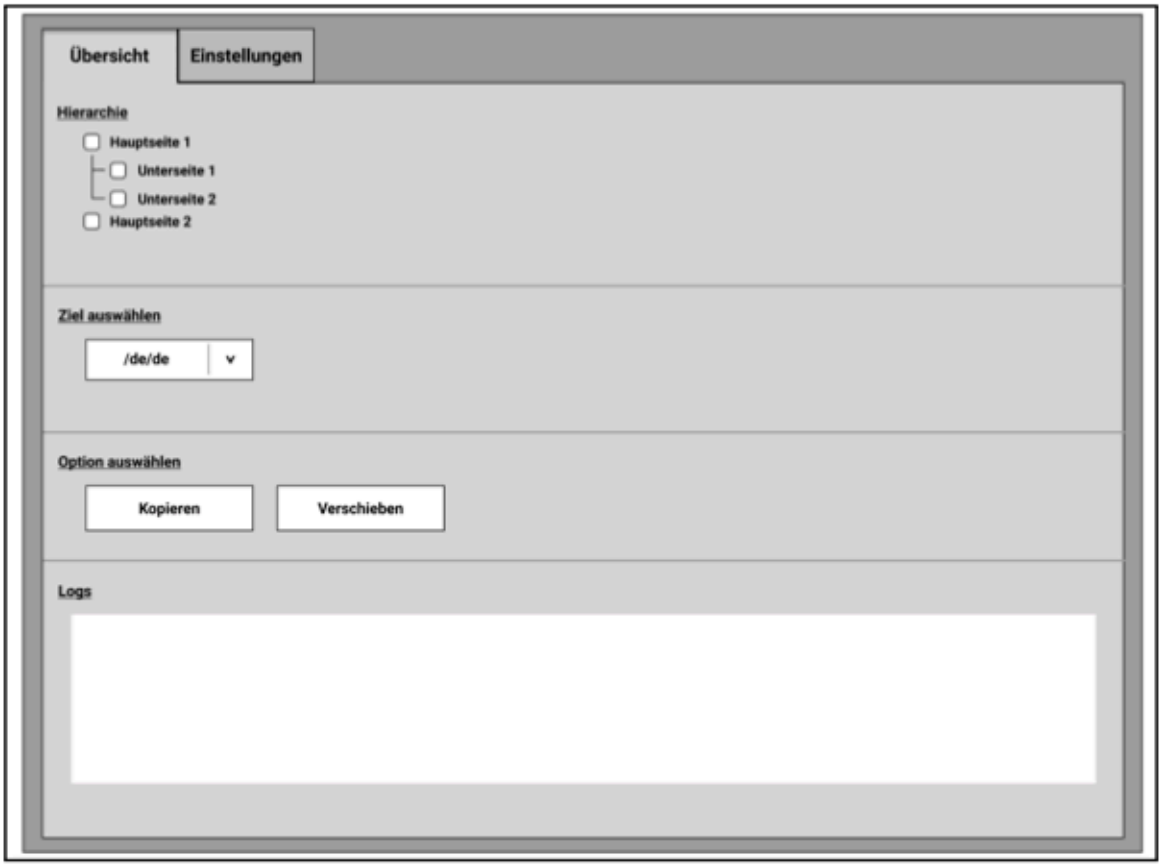

Abbildung 5.13: Das Fenster „Übersicht“ der Plugin-Einstellungen

Die Navigationsleiste in Abbildung 5.13 zeigt die Menüpunkte „Übersicht“ und „Einstellungen“. In den folgenden Abschnitten werden zunächst die Elemente des ersten Menüpunktes näher erläutert.

5.2.1 Übersicht

Der obere Teil des Inhaltes zeigt ein baumartiges Modell aller Haupt- und Unterseiten (vergleiche 5.13). Jeder Seite wird ein Kontrollkästchen zugeordnet. Dieses kann durch einen Klick mit der Maus ausgewählt werden (siehe 5.14). Wählt ein Nutzer eine Seite aus, die in der Hierarchie darunterliegende Seiten besitzt, so werden diese ebenfalls markiert.

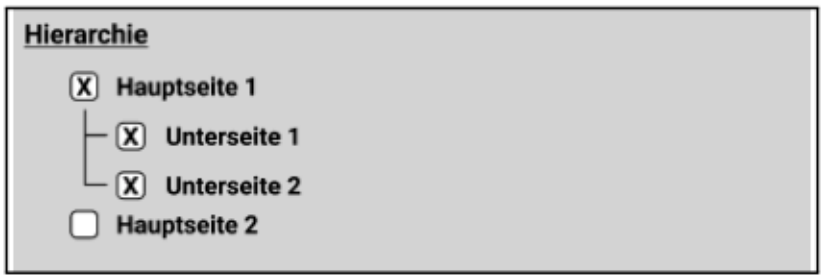

Abbildung 5.14: Ausgewählte Haupt- und Unterseiten

Anschließend muss die Sprache gewählt werden, zu deren Präfix die ausgewählten Haupt- und Unterseiten kopiert oder verschoben werden sollen (vergleiche 5.15). Dies

geschieht über eine Dropdown-Liste (siehe 5.16). Das jeweils ausgewählte Listenelement wird dem Nutzer vor der Bestätigung per Mausklick durch einen Mouse-Hover-Effekt angezeigt (siehe 5.17). Nach der Bestätigung wird die ausgewählte Sprache im Auswahl-Element angezeigt (vergleiche 5.18). Die Sprachenpräfixe sollen aus einer Kombination von „ISO 639-1“ Sprachcodes [49] und einem Code des „ISO 3166-1 alpha-2“-Standards [48] der jeweiligen Region entstehen. Der erste Teil des „ISO 639-1“-Standards, „ISO 639-1:2002“ liefert einen 2-Buchstaben-Identifikations-Code der wichtigsten Weltsprachen [49]. Ein Beispiel für eine solche Kombination aus Sprache und Region ist „en-US“. Diese sollen im Kopier- oder Verschiebevorgang als Präfix im Pfad der erstellten Haupt- und Unterseiten enthalten sein. Zu diesem Zweck werden die Kombinationen aus Sprache und Region in das Format <*Sprache*>/<*Region*> umgewandelt. Eine Beispiel für eine vollständige URL nach diesem Schema wäre *https://url.com/en/us/zieldokument.html*

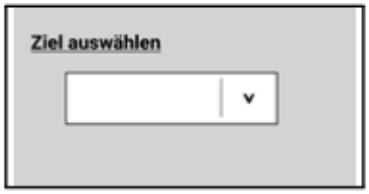

Abbildung 5.15: Auswahl-Element

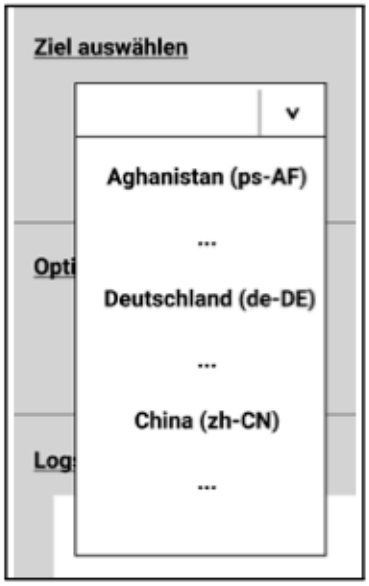

Abbildung 5.16: Dropdown-Liste mit Sprachen

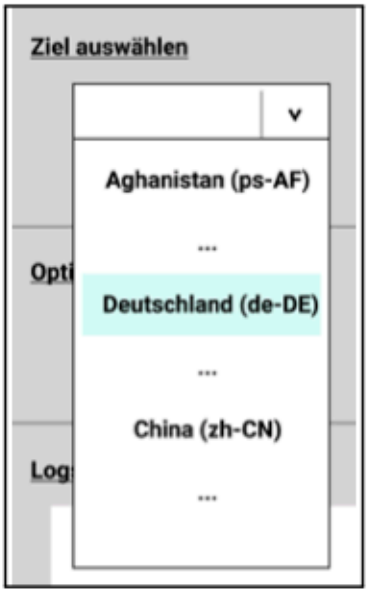

Abbildung 5.17: Hover-Effekt

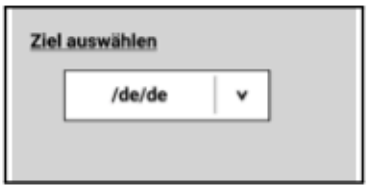

Abbildung 5.18: Auswahl-Element mit getroffener Auswahl

Im Anschluss muss die Aktion ausgewählt werden, die ausgeführt werden soll. Abbildung 5.19 zeigt ein Beispiel eines solchen Auswahlmenüs. Seine Knopfelemente besitzen zur besseren Bedienung ebenfalls einen Mouse-Hover-Effekt.

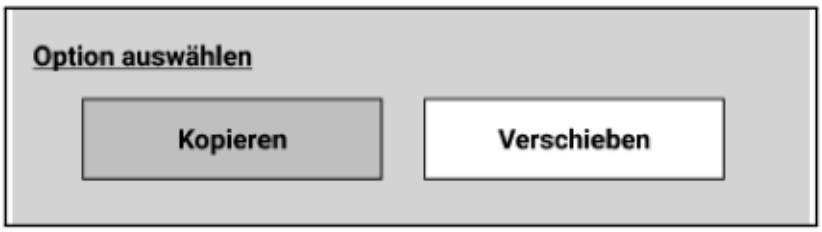

Abbildung 5.19: „Kopieren“ -Knopf mit Hover-Effekt

Nach der Ausführung einer Aktion werden entsprechende Erfolgs- und Fehlermeldungen in einem Protokoll-Fenster angezeigt (siehe 5.20).

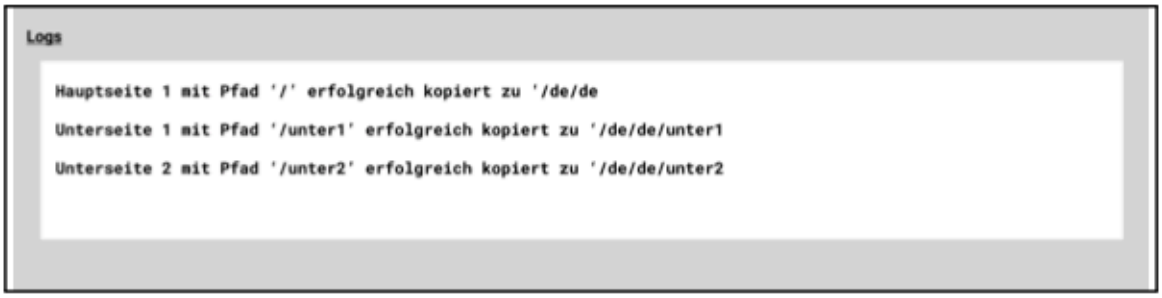

Abbildung 5.20: Meldungen nach erfolgreichem Kopiervorgang

5.2.2 Einstellungen

Im Einstellungsfenster ist es möglich, Attribute von Widgets in ein spezielles Sprachformat zu exportieren. Diese können – nach entsprechender Änderung – auch wieder importiert werden. Daraufhin werden die Texte der Seite mit den importierten Textstellen ersetzt. Weiterhin besteht die Möglichkeit, die Hauptseite der Webseite durch das Plugin verwalten zu lassen. Abbildung 5.21 zeigt ein Modell des Einstellungsfensters.

Abbildung 5.21: Der Einstellungs-Tab

Im oberen Drittel befinden sich die Funktionsknöpfe zum Ex- und Importieren. Betätigt man den Knopf „Exportieren“ , wird eine Datei mit den entsprechenden Daten heruntergeladen. Bei Knopfdruck auf „Importieren“ wird ein durch den Browser erzeugtes Fenster zur Auswahl einer ensptrechenden Datei aufgerufen. Unter den Funktionsknöpfen befindet sich ein Kontrollkästchen zur Verwaltung der Einstellungen der Hauptseite beim initialen Seitenbesuch. Ist dieses auf „Aktiv“ gesetzt, wird die Seite, die beim Laden initial angezeigt wird, je nach den Spracheinstellungen des Browsers automatisch ausgewählt. Das untere Drittel der Abbildung zeigt ein Textfenster, welches Meldungen anzeigt, die bei der Benutzung der vorher genannten Funktionen entstehen. Hier werden Erfolgs und/oder Fehlermeldungen angezeigt (siehe Abbildung 5.22 und Abbildung 5.23).

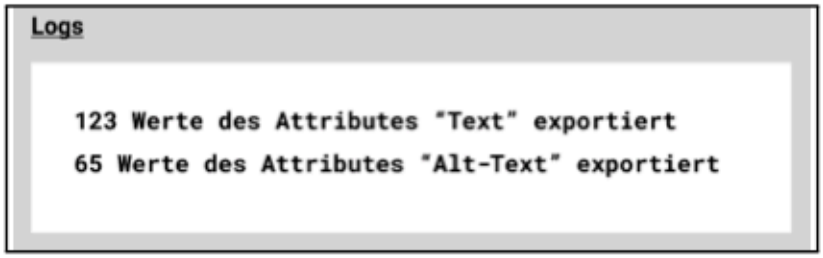

Abbildung 5.22: Erfolgsmeldungen beim Exportieren von Widget-Attributen

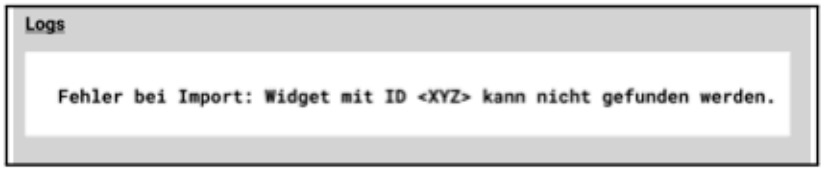

Abbildung 5.23: Fehlermeldung im Importvorgang

6 Implementierung

Nach der Darstellung der nötigen Grundlagen in Kapitel 2 wird anschließend die Umsetzung der in Kapitel 3 und 5 erläuterten Anforderungen und Konzepte dargestellt. Ziel dieses Kapitels ist es, dem Leser Implementierungsdetails und -entscheidungen zugänglich und verständlich vorzustellen. Weiterhin werden die Schritte, die zum letztendlichen Ergebnis der Thesis geführt haben, aufgezeigt. Im Anschluss des Kapitels sollte der Leser in der Lage sein, die benötigte Entwicklungsumgebung einzurichten den Quelltext der Anwendung fehlerfrei auszuführen[1].

6.1 Einrichten der Entwicklungsumgebung

6.1.1 Betriebssystem

Die Implementierung zur Lösung der Problemstellung dieser Ausarbeitung wurde auf dem Betriebssystem „MacOS Catalina“ vollzogen. Da es sich bei MacOS um ein UNIX-basiertes Betriebssystem handelt [74], sollte die Ausführung auf anderen UNIX-basierten Betriebssystemen ebenfalls problemlos möglich sein. Die Implementierung dieser Ausarbeitung wurde nicht auf einem Windows-basierten Betriebssystems getestet, weswegen die korrekte Funktionalität des Programmcodes auf einem solchen nicht gewährleistet werden kann.

6.1.2 Code-Editor

Zur Entwicklung, Verwaltung und Aufbereitung des Programmcodes wurde der frei erhältliche, quelloffene Code-Editor „Visual Studio Code“ (auch: „VS Code“) verwendet [108]. VS Code erleichtert die Entwicklung von Quelltext durch eine automatische Vervollständigung von Funktions-, Klassen- und Variablennamen. Weiterhin liefert VS Code die Darstellung von Programmcode mithilfe einer farblichen Hervorhebung der Syntax einer Programmiersprache. Eine breit aufgestellte Palette an Erweiterungen bietet zusätzliche Funktionalität.

6.1.3 Erforderliche Software und Installation

Zur Installation der Programmteile wird eine Kommandozeilenumgebung in einem Terminal benötigt. Weiterhin muss auf dem Zielsystem die Software „git“ installiert sein, um

[1] Unter der Vorraussetzung, dass er über einen Scrivito-Account und eine auf „my.scrivito.com“ erstelle Webseite verfügt

den Quellcode aus dem unten angegebenen Git-Repositorium herunterzuladen. Dieser Vorgang wird mit dem folgenden Shell-Kommando ausgelöst.

```
git clone [redacted]
```

Der damit erzeugte Ordner wird im Folgenden als „Hauptverzeichnis“bezeichnet. Die zur Ausführung benötigten zusätzlichen Bibliotheken (Abhängigkeiten) befinden sich in der Datei `package.json` im Hauptverzeichnis des Projektpaketes. Um diese auf das System zu laden, wird die Installation der Kommandozeilenschnittstelle des „Node Package Managers“ (npm) vorausgesetzt [73]. Nach dem Kopieren des Quelltextes mit git muss zunächst in den entsprechenden Ordner gewechselt werden. Die Installation der Abhängigkeiten wird daraufhin mit folgendem Befehl ausgelöst.

```
cd <Hauptverzeichnis>
npm install
```

6.1.4 Kompilierung und Programmstart

Um den in den letzten beiden Schritten beschriebenen Installationsvorgang abzuschließen, muss der Quelltext nun kompiliert werden. Das geschieht über Kommandozeilenparameter, die ebenfalls in der Datei `package.json` definiert sind. Die erzeugte Version des Programmcodes liegt nach diesem Arbeitsschritt im `build`-Ordner des Hauptverzeichnisses. Die erzeugten Dateien können dann gegebenenfalls auf einem Web-Server oder Cloud-Hosting-Provider abgelegt werden.

```
npm run build
```

Möchte man die Laufzeitumgebung stattdessen lokal starten, muss der Shell-Befehl zur Kompilierung nicht explizit aufgerufen werden.

```
npm run start
```

Nach dem Ausführen dieses Kommandos startet über Create-React-App ein lokaler Entwicklungsserver (siehe 2.5.3). In diesem Schritt werden die erzeugten Dateien nicht im `build`-Ordner abgelegt, sondern in den RAM-Speicher des Computers geladen. Nach dem Kompilierungsschritt öffnet sich der vom Nutzer festgelegte Standard-Browser. Die lokale Internetadresse der auf dem Webpack-Server bereitgestellten Instanz des Quellcodes wird automatisch angewählt. Das Programm kann nun lokal verwendet werden.

6.2 Verwendete Programmiersprachen und Frameworks

In den folgenden Unterkapiteln wird die Verwendung Programmiersprachen und Frameworks, die in der der Implementierung dieser Ausarbeitung verwendet wurden, anhand von Beispielen erläutert. Die dazugehörigen Grundlagen und Verwendungszwecke wurden bereits in Kapitel 2 dargestellt.

React

Zur Entwicklung der Benutzeroberflächen wurde React verwendet. Die Entwicklungsumgebung der Implementierung wurde zunächst mit dem Kommandozeilenwerkzeug „create-react-app“ erstellt. Dieses liefert ein vorkonfiguriertes Grundgerüst einer React-Anwendung.

Wie in 2.5.2 beschrieben, liefert React zwei unterschiedliche Wege, Komponenten zu definieren. Beide sind in der Implementierung an unterschiedlichen Stellen zu finden.

Klassenbasierte Komponenten sind wie folgt definiert.

```
class MyComponent extends React.Component {

  constructor(props){
      super(props);
  }

  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Funktionsbasierte Komponenten können in folgender Schreibweise ausgedrückt werden.

```
const FunctionComponent = (props) => {
    return (
        <h1>Hello, {props.name}</h1>;
    )
}
```

SCSS

Die meisten Stilregeln der Implementierung befinden sich in Dateien mit der Endung `*.module.scss`. Diese Art von Datei stellt eine Kombination aus den in den Kapitel 2 erläuterten Varianten CSS-Module und SCSS dar. Das erlaubt das Verwenden von geschachtelten Stilregeln und SCSS-Funktionen innerhalb der Style-Dateien zusammen mit der Möglichkeit, Stilregeln voneinander abzukapseln.

```
// Importieren von Variablen
@import "../../scss/globals-colors";

// Styles for Navbar
.beispiel1 {
    color: $farbe1;

    // geschachtelter Klassenname
    .beispiel1__inhalt {
```

```
        color: $farbe2;
    }
}

.beispiel2 {
    color: green;
}
```

SCSS-Dateien werden dann im Compile-Prozess in CSS-Dateien umgewandelt.

Listing 6.1: SCSS Quelltext [83]

```
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }

    li {
        display: inline-block;
    }

    a {
        display: block;
        padding: 6px 12px;
        text-decoration: none;
    }
}
```

->

Listing 6.2: Erzeugter CSS Quelltext [83]

```
nav ul {
    margin: 0;
    padding: 0;
    list-style: none;
}
nav li {
    display: inline-block;
}
nav a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
}
```

Um CSS-Dateien in das Projekt einzubinden und dem Module-Bundler Webpack bekannt zu machen, müssen diese innerhalb einer JavaScript-Datei importiert werden. Das ist unter anderem auf die beiden, hier vorgestellten Arten möglich.

```
// Import mit Seiteneffekten. Alle Klassennamen werden im Browser im globalen
    Namensraum verfügbar gemacht.
import './beispiel.module.scss';

// Alle Klassennamen sind, sofern nicht mit ,,:global“ angegeben, nur in dieser
    Datei innerhalb eines Objekt mit dem Namen ,,style“ vorhanden.
import styles from './beispiel.module.scss'

//beispiel1 ist ein globaler Klassenname
const Beispiel1 = (<h1 className='beispiel1'>
                    </h1>;

//beispiel2 ist ein lokaler, eindeutiger Klassenname
const Beispiel2 = (<h1 className={styles.beispiel2}>
                    </h1>;
```

6.3 TypeScript-Konfiguration

Um TypeScript zu konfigurieren, wird eine Datei mit dem Namen `tsconfig.json` im Hauptverzeichnis der Ordnerstruktur benötigt. Die in dieser Ausarbeitung verwendete `tsconfig.json` wurde automatisiert durch create-react-app erstellt.

Um CSS-Module leichter benutzen zu können, wurde das VSCode Plugin „CSS-Modules“ eingebunden [65]. Dieses erweitert die automatische Vervollständigung eines Code-Editors beim Zugriff auf ein CSS-Modul-Objekt um alle vorhandenen Klassennamen. Außerdem ist es nötig, die TypeScript-Konfigurationsdatei mit dem Plugin „typescript-plugin-css-modules“ zu ergänzen, um zu CSS-Modul-Dateien Typendefinitionen zu generieren, da der `tsc`-Compiler sonst die Nutzung der CSS-Module als fehlerhaft markiert.

```
// Styles for BackgroundImageWidget

.background_image_widget {
    background-color: white;
}
```

Abbildung 6.1: Definition einer CSS-Klasse innerhalb einer CSS-Modul-Datei

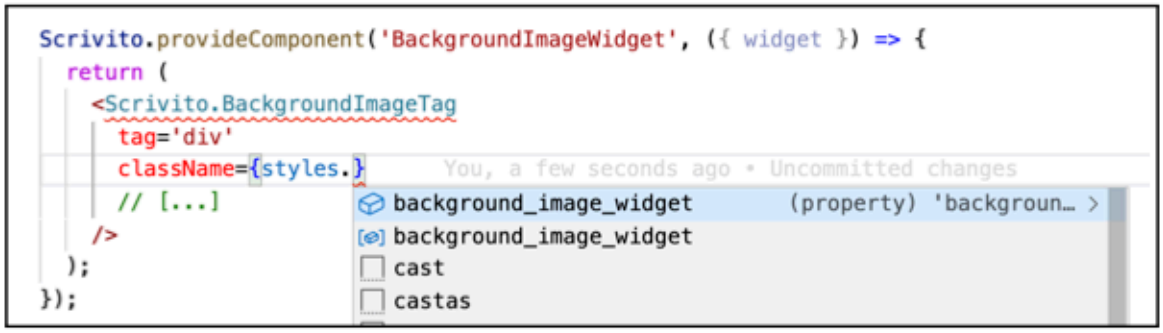

Abbildung 6.2: Objekt Zugriff auf das CSS-Modul im Code-Editor

6.4 Entwicklung von TypeScript-Definitionen für Scrivito

Scrivito liefert zur Entwicklung die sogenannte „Scrivito-SDK“ als JavaScript-Modul [68]. Zu Beginn dieser Ausarbeitung existierten noch keine Typendefinitionen für dieses Paket auf *DefinitelyTyped*. Das Scrivito-Modul war somit zwar mit TypeScript-Dateien kompatibel, lieferte aber keine Typeninformationen über Parameter und Rückgabewerte seiner Funktionen. Deswegen wurden im Rahmen dieser Ausarbeitung quelloffene Typendefinitionen für Scrivito nachentwickelt. In Absprache mit einem Scrivito-Entwickler konnten die Typendefinitionen vom Scrivito-Team überprüft und anschließend veröffentlich werden [57].

6.5 Umsetzung der „[redacted]“-Designvorgaben

Wie bereits in 5.1.2 erläutert, wurden die zur Umsetzung nötigen Designvorgaben in Kontrollelemente, Bausteine und Ansichten unterteilt. In Scrivito sind diese über Widgets umsetzbar.

Zur Verständnis wird dieser Vorgang anhand eines „Image-Widget“-Kontrollelementes erklärt. Bei diesem handelt es sich wiederverwendbares Element, welches ein Bild darstellt und verschiedene Konfigurationsmöglichkeiten bietet. Um ein Widget in Scrivito umzusetzen, werden vier Bestandteile benötigt. Jeder Bestandteil wird innerhalb einer Datei definiert. Dieser Vorgang wird innerhalb der nächsten vier Unterkapitel dargestellt.

6.5.1 Anlegen einer Widget-Klasse

Als erstes muss zu einem Widget eine `WidgetClass` erstellt werden. Diese teilt ihm einen Namen, Attribute und weitere optionale Konfigurationsparamter zu. Für diesen Vorgang bietet Scrivito die Funktion `provideWidgetClass`. Im Fall des Image-Widgets wird diese in der Datei `ImageWidgetClass.ts` eingesetzt (siehe 6.3).

Listing 6.3: ImageWidgetClass.ts

```
import * as Scrivito from 'scrivito';

const ImageWidgetClass: WidgetClass = Scrivito.provideWidgetClass('
    ImageWidget', {
  attributes: {
    image: 'reference',
    sizing: ['enum', { values: ['imagesize', 'containersize'] }]
  }
});

export default ImageWidgetClass;
```

Um auf die Funktion zugreifen zu können, muss vor ihrer Verwendung das Scrivito-Modul in die entsprechende Datei eingebunden werden. Ihr erster Funktionsparameter bestimmt den Namen des Widgets. Der nächste liefert ein JavaScript-Objekt mit dem Schlüssel `attributes`. Dessen Objektschlüssel bilden die Namen der Attribute des Widgets. Die Werte der Objektschlüssel beschreiben den Typ des jeweiligen Attributes. Scrivito bietet dafür folgende Attribute:

- `date`: Ein Datum ohne Uhrzeit
- `datetime`: Ein Datum mit Uhrzeit
- `enum`: Eine Aufzählung von Werten. Es kann maximal ein Wert ausgewählt werden.
- `multienum`: Eine Aufzählung von Werten. Es können einer, mehrere oder keine Werte ausgewählt sein.

- `html`: HTML-Quelltext
- `string`: Eine Zeichenkette
- `stringlist`: Eine Liste von Zeichenketten
- `integer`: Eine ganze Zahl
- `float`: Eine Gleitkommazahl
- `binary`: Ein Puffer mit Binärdaten
- `link`: Eine Verlinkung auf externe oder interne Web- oder Unterseiten
- `reference`: Eine Referenz auf ein anderes, enthaltenes Scrivito-Objekt
- `referencelist`: Eine Liste von Referenzen auf andere, enthaltene Scrivito-Objekte
- `widgetlist`: Eine Liste von enthaltenen Widget-Instanzen

Das Image-Widget in Abbildung 6.3 verfügt über zwei Attribute. Ein `image`-Attribut zum Speichern einer Referenz auf ein Bild und ein `sizing`-Attribut mit Werten, die sich auf die dargestellte Größe des Bildes beziehen. Da es sich bei `sizing` um ein Enum-Attribut handelt, kann entweder der Wert `imagesize` oder `containersize` ausgewählt werden. Ersterer stellt die Größe des Image-Widgets auf die tatsächliche Größe des Bildes ein. Ist ein Bild zu groß, kann es mit der Einstellung `containersize` auf die Größe des umliegenden Container-Elementes geschrumpft werden.

Der Rückgabe-Wert von `provideWidgetClass` ist eine JavaScript-Klasse. Diese liefert Scrivito alle Informationen, die zur Instanziierung eines Widgets gebraucht werden. Damit die Widget-Klasse `ImageWidgetClass` nicht nur von Scrivito, sondern auch programmatisch genutzt werden kann, wird sie am Ende der Datei exportiert. Damit kann sie auch in anderen JavaScript-Dateien eingebunden werden.

6.5.2 Festlegen einer Editor-Übersicht

Um ein Widget über Scrivitos Drag-And-Drop-System verfügbar und im WYSIWYG-Editor konfigurierbar zu machen, muss diesem eine sogenannte `EditingConfig` zugewiesen werden. Diese legt unter anderem den Titel, die Beschreibung und eine Miniaturansicht des Widgets im Widget-Auswahl-Menü fest. Weiterhin ist es möglich, in der `EditingConfig` die Editor-Übersicht eines Widgets zu konfigurieren. Zugriff darauf hat man im Options-Menü eines Widgets (siehe 6.3). Dazu stehen bei der Festlegung verschiedene Parameter zur Verfügung. Im Rahmen des Image-Widgets wird die für diesen Vorgang verwendete Funktion `provideEditingConfig` in der Datei `ImageWidgetEditingConfig.ts` genutzt (siehe 6.4).

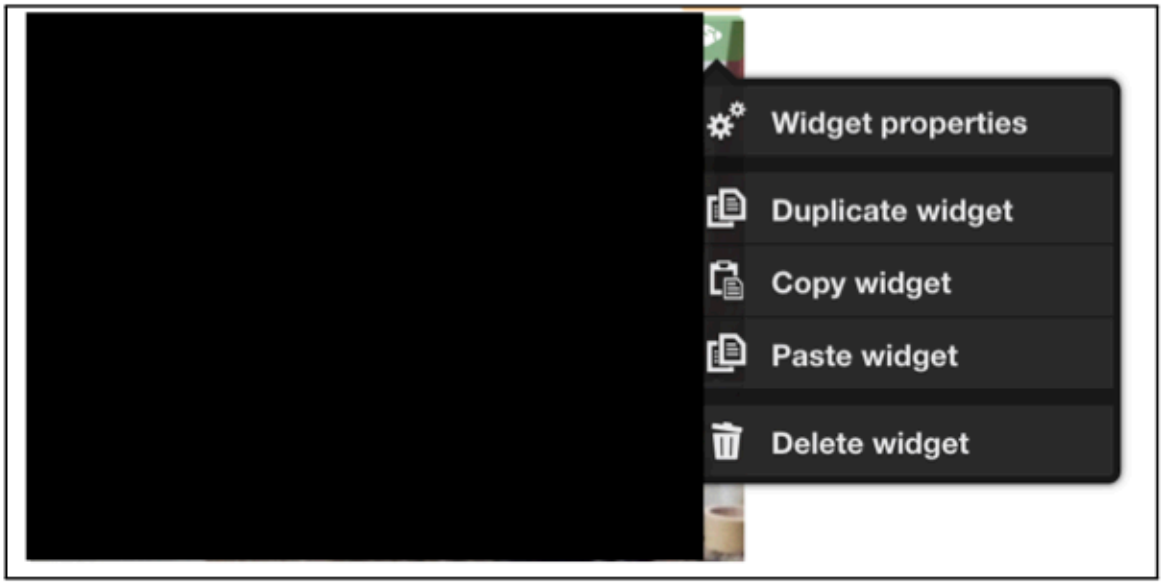

Abbildung 6.3: Zugriff auf das Editoren-Menü eines Widgets

Listing 6.4: ImageWidgetEditingConfig.ts

```
import * as Scrivito from 'scrivito';

Scrivito.provideEditingConfig('ImageWidget', {
  title: 'Image Widget',
  attributes: {
    sizing: {
      title: 'Sizing of the image',
      values: [
        { value: 'imagesize', title: 'Image Size' },
        { value: 'containersize', title: 'Container Size' }
      ]
    }
  },
  properties: ['sizing']
});
```

Ähnlich zu dem in Abbildung 6.3 gezeigten Beispiel muss das Scrivito-Modul auch hier eingebunden werden. Der erste Parameter von `provideEditingConfig` beschreibt den Namen des Widgets. Im zweiten Funktionsargument werden unter anderem Konfigurationsparameter der Editor-Übersicht geliefert. Die in den Feldern `attributes` und `properties` angegebenen Werte beziehen sich auf die in 6.3 festgelegten Attribute des Widgets. Konkret verfügt dieses über die Attribute `image` und `sizing`. Scrivito liefert bei einigen Attribut-Typen die Möglichkeit, den Wert unmittelbar in seinem WYSIWYG-Editor (also nicht über die Editor-Übersicht) anzupassen. Dazu muss das Attribut in der Editor-Konfiguration ausgelassen werden. Das ist bei dem *image*-Attribut der Fall. Über das Feld `sizing` wird das gleichnamige Attribut des Image-Widgets für die Editoren-Ansicht konfiguriert. Es beinhaltet neben dessen Titel auch ein Feld `values`. Dieses legt für die Werte, die `sizing`

annehmen kann, jeweils einen Titel fest. Außerdem wird über das Feld `properties` angegeben, dass `sizing` in den Hauptreiter der Editoren-Ansicht des Image-Widgets aufgenommen werden soll (siehe 6.4).

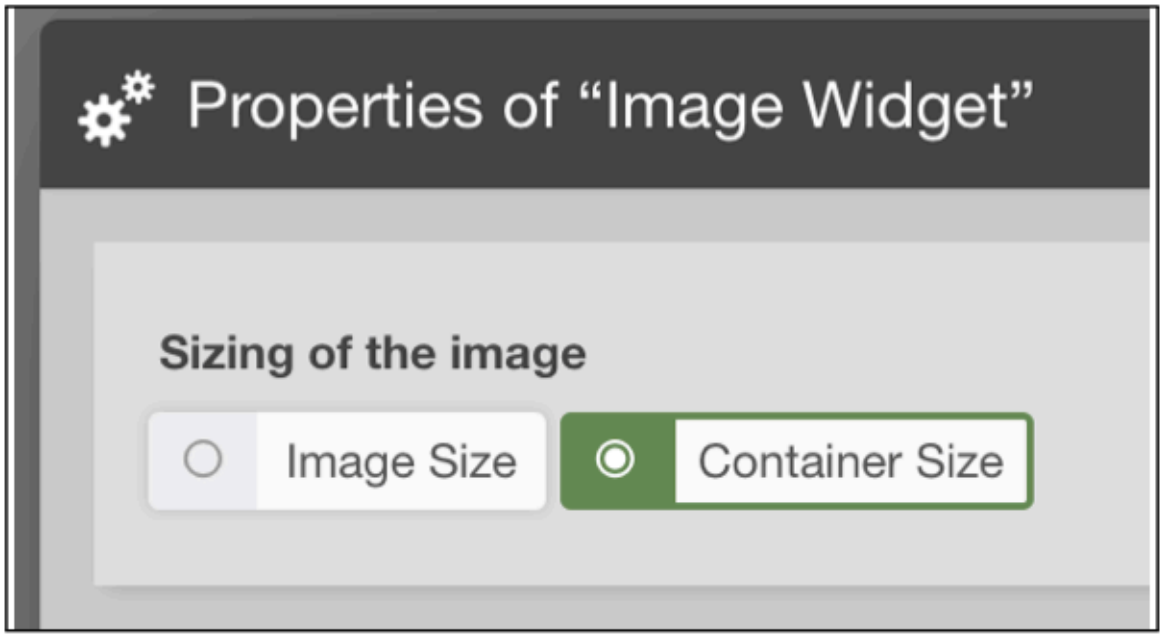

Abbildung 6.4: Einstellungen des Image-Widgets

6.5.3 Definieren einer React-Komponente

Damit das Image-Widget im WYSIWYG-Editor dargestellt werden kann, benötigt es eine React-Komponente. Diese wird in der Datei `ImageWidgetComponent.tsx` festgelegt. Dazu bietet Scrivito die Funktion `provideComponent` (siehe 6.5).

Listing 6.5: ImageWidgetComponent.tsx

```
import * as React from 'react';
import * as Scrivito from 'scrivito';
import styles from './ImageWidget.module.scss';

interface ImageWidgetProps {
  widget: Scrivito.Widget
}

const ImageWidgetComponent = (props: ImageWidgetProps) => {

  const { widget } = props;

  const sizing = widget.get('sizing') === 'containersize';

  const className = sizing ? styles.container_size : undefined;

  return <Scrivito.ImageTag
```

```
    content={widget}
    className={className}
    attribute='image'
  />;
};

Scrivito.provideComponent('ImageWidget', ImageWidgetComponent);
```

Das erste Funktionsargument von `provideComponent` ist, wie in 6.3 und 6.4, der Name des Widgets. Der zweite Funktionsparameter ist eine funktionale React-Komponente mit dem Namen `ImageWidgetComponent`. Durch die Verknüpfung der React-Komponenten und des Image-Widgets über die `provideComponent`-Funktion ist Scrivito in der Lage, bei der Verwendung des Image-Widgets eine Instanz seiner Widget-Klasse an die React-Komponente zu übergeben. Dieser Übergabeparameter wird im Interface der `props` von ImageWidgetComponent angegeben. Durch ihn gelangt die React-Komponente an die Attribute ihrer Widget-Instanz, in diesem Fall `sizing`. Die Darstellung eines Image-Widgets wird durch den Rückgabewert von `ImageWidgetComponent` bestimmt. Dieser bezeichnet eine `ImageTag`-Komponente des Scrivito-Moduls. Sie benötigt den Parameter `content` mit der Widget-Instanz des Image-Widgets als Wert und den Parameter `attribute`. Letzterer beziffert den Namen des Attributes der Image-Widget-Klasse, die das darzustellende Bild enthält. Zusätzlich kann an `ImageTag` ein `className`-Parameter übergeben werden. Ist das Interaktionselement von `sizing` der Image-Widget-Editorenübersicht auf `Container Size` gesetzt, so wird über das `sizing` Attribut der Wert `containersize` an die React-Komponente übergeben. Anschließend wird über das `styles`-Objekt in der ImageTag-Komponente der entsprechende Klassenname zur Umsetzung dieser Funktion gesetzt (siehe 6.6).

6.5.4 Anlegen einer Stylesheet-Datei

Die Darstellungsparameter einer Komponente werden über Klassennamen in CSS-Modulen angegeben. In der Image-Widget-Komponente geschieht dies über die Datei `ImageWidget.module.scss` (siehe 6.6).

Listing 6.6: ImageWidget.module.scss

```
.container_size {
    max-width: 100%;
}
```

Das Image-Widget benötigt nur eine Regel. Wie in Abbildung 6.5 gezeigt, wird der Klassenname `container_size` nur dann auf die von `ImageWidgetComponent` zurückgegebene ImageTag-Komponente gesetzt, wenn das `sizing`-Attribut einer Image-Widget-Instanz auf `containersize` gesetzt ist. Der CSS-Klassenname legt dann fest, dass `ImageWidgetComponent` und sein enthaltenes Bild höchstens die Größe des umliegenden Elementes annehmen dürfen.

6.5.5 Zusammenfassung

Der Vorgang der Erstellung verschiedener Scrivito-Widgets unterscheidet sich im wesentlichen von der Implementierung des Image-Widgets in der Anzahl ihrer Attribute und der daraus entstehenden Komplexität. Jeder der in Kapitel 5.1.2 erwähnten Kontrollelemente, Bausteine und Ansichten durchlief in der Anfertigung dieser Ausarbeitung einen Prozess zur Umwandlung in ein Scrivito-Widget, der dem in den Abschnitten 6.5.1 – 6.5.4 dargestellten Prozess äquivalent ist. Handelte es sich nicht um ein Kontrollelement, sondern um einen Baustein oder eine Ansicht, so wurden dessen Widget-Attribute zusätzlich um Attribute des Types `widgetlist` erweitert, die die benötigten Kontrollelemente und Bausteine beinhalten.

6.6 Internationalisierungs-Plugin

6.6.1 Installation

Das Internationalisierungsplugin ist als JavaScript-Paket verfügbar. Es kann wie folgt installiert werden:

```
npm install @juliankrieger/scrivito-i18n-plugin
```

Anschließend muss es nur noch in das Projekt eingebunden werden.

```
import('@juliankrieger/scrivito-i18n-plugin');
```

6.6.2 Anbindung der Internationalisierungs-Funktion an Widgets

Um die vom Plugin verwaltete Internationalisierungs-Funktion an ein beliebiges Widget anzubinden, werden zwei unterschiedliche Funktionen geliefert. Diese müssen bei der Definition des Widgets, also in seiner Widget-Klasse und seiner Editoren-Konfiguration angewandt werden.

Definition der Widget-Klasse Im Gegensatz zur herkömmlichen Definition einer Widget-Klasse müssen die Konfigurationsparameter hier über die Funktion `withI18NWidgetProps` an Scrivito übergeben werden.

Listing 6.7: Ein beliebiges Scrivito-Widget

```
import * as Scrivito from 'scrivito';
import { withI18NWidgetProps } from '@juliankrieger/scrivito-i18n-
    plugin';
const I18NTextWidgetClass = Scrivito.provideWidgetClass('
    I18NTextWidget', withI18NWidgetProps({
  attributes: {
    text: 'html'
  }
```

```
}));

export default I18NTextWidgetClass;
```

Diese erweitert die Konfigurationsparameter zusätzlich um die Attribute `langID`, `i18nEnabled` und `i18nID`. Ersteres stellt das Kürzel einer Sprache dar, mit dem diese im Plugin-System identifiziert wird. Über das Attribut `i18nEnabled` kann das Plugin Widgets erkennen, die in das Internationalisierungssystem eingegliedert sind. Der letzte Parameter stellt eine Identifikationsnummer dar, die zwei oder mehr Widgets miteinander verbindet. Wurden beispielsweise eine Seite und ihre Widgets an ein anderes Sprachpräfix kopiert, kann über die `i18nID` eine Beziehung zum ursprünglichen Widget hergestellt werden. Zusätzlich werden durch die `withI18NWidgetProps`-Funktion die Namen aller Attribute, die im eigentlichen Konfigurationsobjekt der Widget-Klasse an Scrivito übergeben werden sollen, in einem Attribut mit dem Namen `i18nExportableAttributes` gespeichert. Dieses wird in der optionalen Export-Funktion des Plugins benötigt.

Definition der Editoren-Übersicht Um die Attribute, die im vorigen Schritt zur Widget-Klasse des zu erweiternden Widgets hinzugefügt wurden, in der Editoren-Ansicht einsehen zu können, muss diese ebenfalls angepasst werden. Das geschieht mit der durch das Plugin gelieferten Funktion `withI18NEditingConfig`.

Listing 6.8: Ertweitern der Editoren-Ansicht mit „withI18NEditingConfig“

```
import * as Scrivito from 'scrivito';
import { withI18NEditingConfig } from '@juliankrieger/scrivito-i18n
   -plugin';

Scrivito.provideEditingConfig('I18NTextWidget',
   withI18NEditingConfig({
  title: 'I18N Enabled Text Widget',
  description: 'A text widget with i18n enabled capabilities',
  properties: ['text'],
  attributes: {
    text: {
      title: 'Text',
    }

  }
}, ['text']));
```

Diese Funktion erweitert die Editoren-Ansicht des Widgets um ein zusätzliches Fenster, in dem die Widget-spezifischen Parameter des Internationalisierungs-Plugins eingesehen und angepasst werden können. Zusätzlich ist es möglich, über einen zweiten Parameter eine Liste von Attributen anzugeben, die standardmäßig bei jedem Widget dieser Art von der Export-Funktion des Plugins berücksichtigt werden.

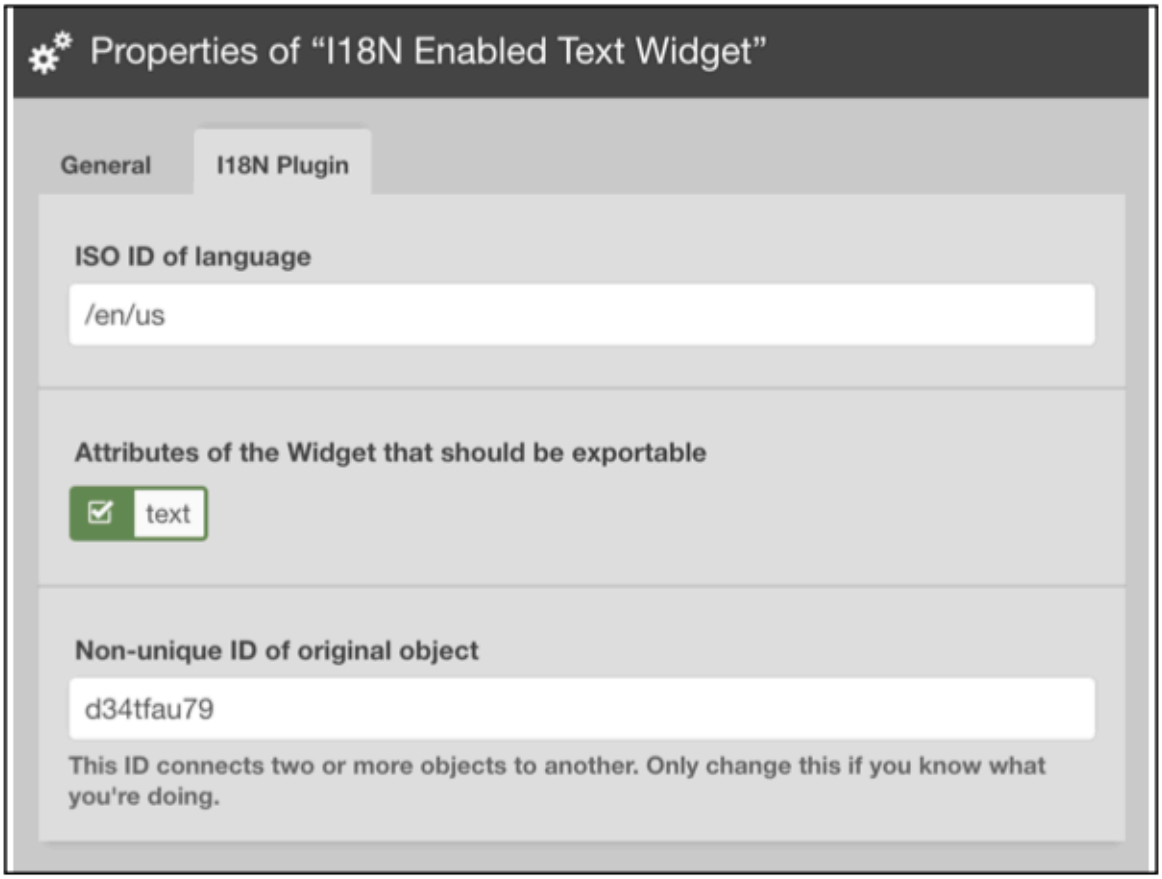

Abbildung 6.5: Plugin-Parameter eines verbundenen Widgets

6.6.3 Benutzeroberfläche

Die Benutzeroberfläche des Plugins ist, wie die Umsetzung der [redacted]-Designvorgaben (siehe 6.5), in React geschrieben. Mit der Scrivito-Funktion `extendMenu` kann das Editoren-Menü in der oberen, rechten Ecke der Scrivito-Benutzeroberfläche um einen Menüpunkt mit dem Titel „i18n Plugin“ erweitert werden. Bei einem Mausklick auf diesen Titel wird ein über das Scrivito-Modul registrierter Dialog geöffnet. Dieser ist dem in 5.2 vorgestellten Modell nachempfunden (siehe 6.6)[2].

[2]Zur besseren Veranschaulichung sind einige Interaktionselemente kompakter dargestellt, als es im wirklichen System der Fall ist

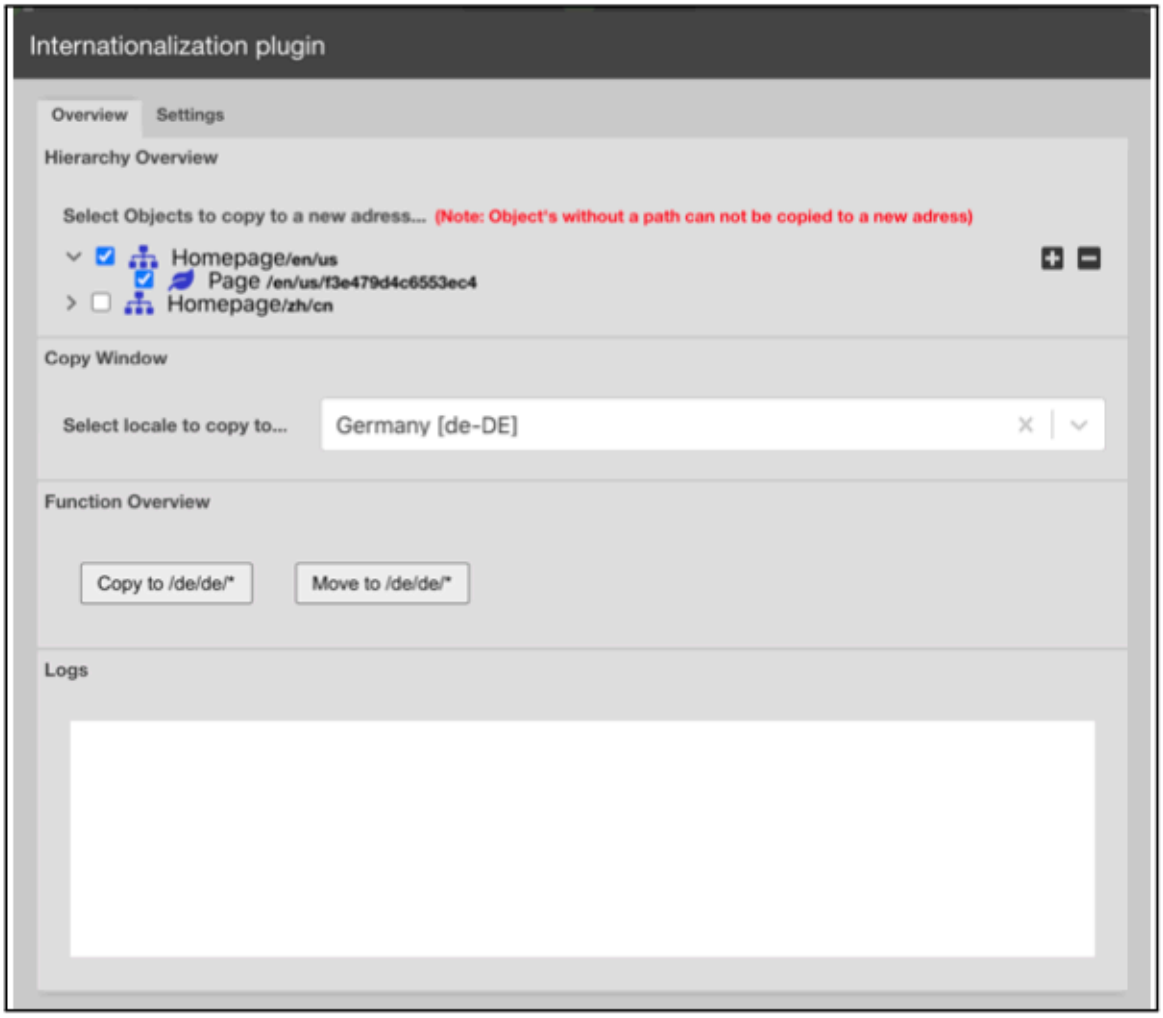

Abbildung 6.6: Internationalisierungs-Plugin: Hauptansicht

Die zur Umsetzung nötige Baumstruktur von Haupt- und Unterseiten wurde mit dem JavaScript-Paket „reactcheckboxtree“erstellt [119]. In Abbildung 6.6 existieren zwei Hauptseiten mit jeweils einer Unterseite: Eine englische und eine chinesische. Wählt man nun eine dieser Seiten aus, kann dies gesamte Hierarchie, nach der Auswahl einer Zielsprache, an das enstprechende Sprachpräfix kopiert oder verschoben werden werden.

6.6.4 Funktionalität des Plugins

Damit das Internationalisierungs-Plugin Parameter wie beispielsweise eine Liste der sprachenabhängigen Haupt- und Unterseiten auch nach dem Neustart der Scrivito-Laufzeit verwalten kann, müssen diese persistiert werden. Um solche Konfigurationseigenschaften in Scrivito persistieren zu können, muss ein sogenanntes „Konfigurationsobjekt“ angelegt werden. Bei diesem handelt es sich um ein Scrivito-Objekt mit zugehöriger Objekt-Klasse und Editor-Konfiguration, welches ausschließlich Daten verwaltet und das über keine Darstellungskomponente verfügt. Um die Funktionalität des Konfigurationsobjektes von Benutzern des Scrivito-Editors abzuschirmen, wird die Editor-Konfiguration des Konfigurationsobjektes bei der Definition um das Attribut `hideInSelectionDialogs` erweitert.

Damit die Existenz des Konfigurationsobjektes zur Laufzeit gewährleistet werden kann, wird es, wenn es nicht vorhanden ist, zur Laufzeit erstellt (siehe 6.7).

Listing 6.9: createConfigAtRuntime.js

```
import * as Scrivito from 'scrivito';

export const configPath = 'i18npluginconfig';

function createConfigAtRuntime () {
  // Warte auf initialisierung von Scrivito
  Scrivito.load(() => {
    /**
     * Gibt ein Objekt zurück, welches das Konfigurationsobjekt
     * und booleansche Werte enthält, die Beschreiben, ob
     * man sich in der Scrivito Benutzeroberfläche befindet.
     */
    return {
      config: Scrivito.Obj.getByPermalink(configPath),
      isEditorLoggedIn: Scrivito.isEditorLoggedIn(),
      isInPlaceEditingActive: Scrivito.isInPlaceEditingActive()
    };
  }).then(result => {
    /**
     * Sind die oben genannten Bedingungen erfüllt, so
     * besteht noch kein Konfigurationsobjekt und
     * es wird eines erstellt.
     */
    if (result && result.isEditorLoggedIn && result.
        isInPlaceEditingActive && !result.config) {
      console.log('Initialising i18n Plugin...');
      Scrivito.getClass('i18nConfig').create({
        _permalink: configPath
      });
    }
  }).catch(err => {
    console.log(err);
  });
}

export default createConfigAtRuntime;
```

Um das Konfigurationsobjekt zu manipulieren, existiert innerhalb des Plugins eine Abstraktion über die gängigsten Methoden. Beispielsweise liefert die Abstraktion eine Funktion zur Verwaltung der Kopier- und Verschiebefunktion in der Benutzeroberfläche. Diese duplizieren oder bearbeiten über Funktionen des Scrivito-Moduls das Scrivito-Objekt aller in der Benutzeroberfläche ausgewählten Seiten und aktualisieren die Parameter der

darin enthaltenen Widgets, die mit dem Plugin kompatibel sind. Die Abstraktion stellt zusätzlich die Funktion zum Export von Internationalisierungsdaten bereit. Die grafische Benutzeroberfläche dafür ähnelt dem in 5.2.2 gezeigten Modell.

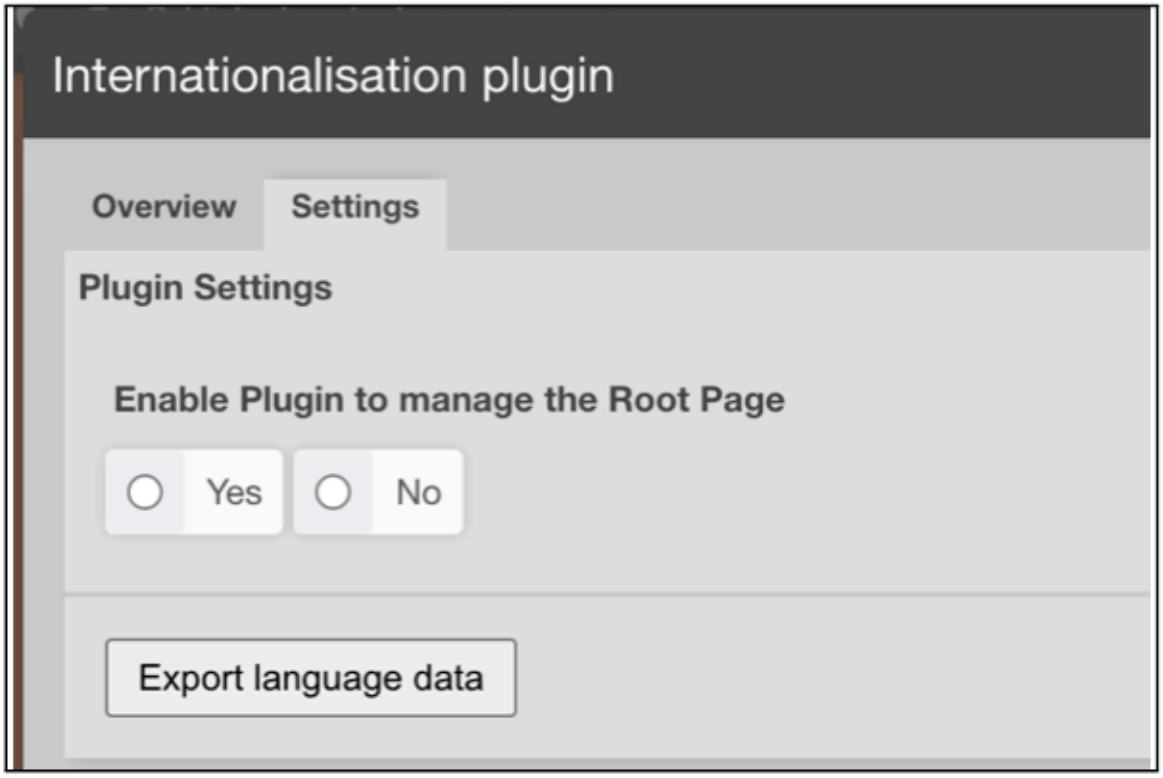

Abbildung 6.7: Internationalisierungs-Plugin: Hauptansicht

Zusätzlich liefert das Plugin eine JavaScript-Funktion, die alle Sprachen und die entsprechenden Pfade zurückgibt, welche im Plugin registriert und damit im Scrivito-System im Einsatz sind. Sie kann beispielsweise zur Erstellung einer Navigationsleiste genutzt werden. Wurde der Kopier- oder Verschiebevorgang erfolgreich abgeschlossen, kann im Scrivito-System anschließend auf die neu erstellte Seite navigiert werden. Dort kann sie dann durch einen Redakteur oder Editor übersetzt werden.

6.7 Test

Im Rahmen dieser Ausarbeitung wurde versucht, sowohl die erarbeitete Umsetzung des Internationalisierungssystems als auch die für die [redacted]-Webseite benötigten Scrivito-Widgets mit dem „Ende-zu-Ende-Testsystem“ „Cypress“ zu testen [54]. Cypress ist in der Lage, die Bedienung eines Interner-Browsers durch einen Nutzer zu simulieren. Danach kann getestet werden, ob die die gewünschte Interaktion korrekt vollzogen wurde und das gewünschtes Ergebnis erreicht wurde [28].

Aufgrund der komplizierten Logik von Scrivito zur Authentifizierung eines Nutzers im CMS konnte eine Cypress Integration auch nach mehreren Anläufen und mit der Unterstützung von mit Cypress erfahrenen Kollegen nicht bewerkstelligt werden. Weiterhin erschwerend ist, dass der Scrivito-Editor über einen sogenannten „Cross-Origin-Iframe“

funktioniert. Cypress ist nicht in der Lage, mit Elementen zu kommunizieren, die sich in einem solchen befinden [20].

7 Evaluation

7.1 Evaluation der erarbeiteten Lösung

In den folgenden Unterkapiteln werden die in dieser Ausarbeitung erarbeiteten Konzepte und Lösungen systematisch begutachtet. Dieser Vorgang ist an den ISO 25010 Standard angelehnt. ISO 25010 ist eine internationale Norm zur Auswertung der Qualität eines Softwaresystems anhand unterschiedlicher Kriterien. [50]

Nachfolgend werden die Implementierung der [redacted]-Webseite und des für Scrivito entwickelten Internationalisierungsplugins anhand dieser Kriterien bewertet.

7.1.1 Funktionale Eignung

In diesem Unterkapitel wird untersucht, inwiefern die in Kapitel 5 und Kapitel 6 konzipierten und implementierten Lösungen den in Kapitel 3 dargestellten Anforderungen entsprechen. Das Maß der funktionalen Eignung lässt sich unter anderem an zwei Merkmalen feststellen. Das erste Merkmal, die sogenannte „funktionale Vollständigkeit", beschreibt, ob und inwiefern die Funktionen der erarbeiteten Lösungen die Anforderungen vollständig abdecken. Das Merkmal der „funktionalen Korrektheit" stellt dar, ob die Lösungen korrekte Ergebnisse bei der Ausführung ihrer Funktionen liefern.

Funktionale Vollständigkeit: Die in 3.1.2 beschriebenen Anforderungen, also die Überführung der Designvorgaben in das Content-Management-System Scrivito wurden größtenteils erfüllt. Alle benötigten Inhalte und Funktionalitäten wurden in Kontrollelemente, Bausteine und Ansichten eingeteilt und in Scrivito-Widgets überführt (siehe 5.1.2). Diese besitzen alle Funktionen und Kontrollparameter, die nötig sind, um die grundlegenden Anforderungen der Designvorgaben umzusetzen. So besitzt beispielsweise ein Textkontrollelement alle Konfigurationseigenschaften, um die benötigten Textfarben, Textgrößen, Schriftarten und Zeilenhöhen oder -abstände festzulegen. Weiterhin wurden die Scrivito-Widgets im Scrivito-Editor eingesetzt und mit entsprechenden Inhalten der englischen Sprache befüllt, sofern sie zum Zeitpunkt der Ausarbeitung bekannt waren. Zur Erstellung von Inhalten in chinesischer Sprache ist ein Meeting mit einer chinesischsprachigen Korrespondentin geplant.

Die in Kapitel 5.2 beschriebenen Funktionen zur Verwaltung von internationalen Haupt-Unterseiten beziehungsweise Inhalten wurden zum Großteil fertiggestellt. Aufgrund des Zeitrahmens von drei Monaten konnte zwar die Logik zum Exportieren von internationalen Texten umgesetzt werden. Die Umsetzung des Gegenstücks – also die Logik zum

Importieren internationaler Texten – war jedoch aus Zeitgründen nicht möglich. Dennoch ist die erarbeitete Lösung in der Lage, internationale Haupt- und Unterseiten anzulegen und zu verwalten.

Funktionale Korrektheit Für komplizierte Funktionen der erarbeiteten Lösung existieren nur wenige Tests. Die Funktionsweise der Scrivito-SDK erschwert es aufgrund verschiedener Eigenschaften, Scrivito-Komponenten zu testen. Es existiert weder eine Dokumentation zu Testverfahren, noch ein vordefiniertes Test-System zur Überprüfung von Scrivito-abhängiger Logik.

Aufgrund der in 6.7 dargestellten Probleme ein Test-Werkzeug einzubinden, konnte die Lösung nicht automatisiert getestet werden. Erschwerend kommt hinzu, dass zum Test des Internationalisierungsplugins Funktionen benötigt werden, welche ohne einen Scrivito-Account nicht verwendet werden können. Die funktionale Korrektheit der Scrivito-Widgets und der Funktionalität der Internationalisierungsfunktion konnten somit nur manuell bestätigt werden.

Die Designvorgaben konnten innerhalb dieser Ausarbeitung weitestgehend korrekt implementiert werden. Um Differenzen zwischen den Designvorgaben und der Umsetzung zu identifizieren, wurde innerhalb von Ergosign ein Review-Prozess mit den am Projekt beteiligten Designern durchgeführt. Die im Design-Review aufgelisteten Diskrepanzen beziehen sich im wesentlichen auf überflüssige oder fehlende Leerräume beziehungsweise auf Fehler in der Darstellung von Texten. Sie wurden bis zum Abgabedatum dieser Ausarbeitung weitestgehend und nach soweit wie möglich ausgebessert.

7.1.2 Effizienz und Performanz

Effizienz und Performanz bei Webseiten und -anwendungen werden zum Beispiel durch einen sogenannten „Lighthouse“-Test charakterisiert [39]. Dieser kann unter anderem mit Hilfe der Entwicklerwerkzeuge des Internet-Browsers „Google Chrome“ berechnet werden (siehe 7.1). Für eine darüber hinausgehende, detailliertere Analyse kann der Test der Webseite „webpagetest.org“ verwendet werden (siehe 7.3 und 7.2) [111]. Eine umfassende Übersicht ist im Anhang unter A.1 und A.2 aufgeführt. Durch den Test von webpagetest.org wurden verschiedene Verbesserungsmöglichkeit in der Entwicklung der [redacted]-Webseite identifiziert. Zum einen wird ein Großteil der Ladezeit durch Anfragen zum Nachladen von Bildern verursacht. Letztere könnten nachträglich durch komprimiertere Versionen ersetzt werden oder über sogenanntes „Lazy-Loading“ genau dann angefragt werden, wenn der Nutzer sich zu ihrer Position bewegt. Das Laden der unkomprimierten Schriftarten (Dateien mit der Endung „.woff2“) verzögert den vollständigen Aufbau der Webseite weiter.

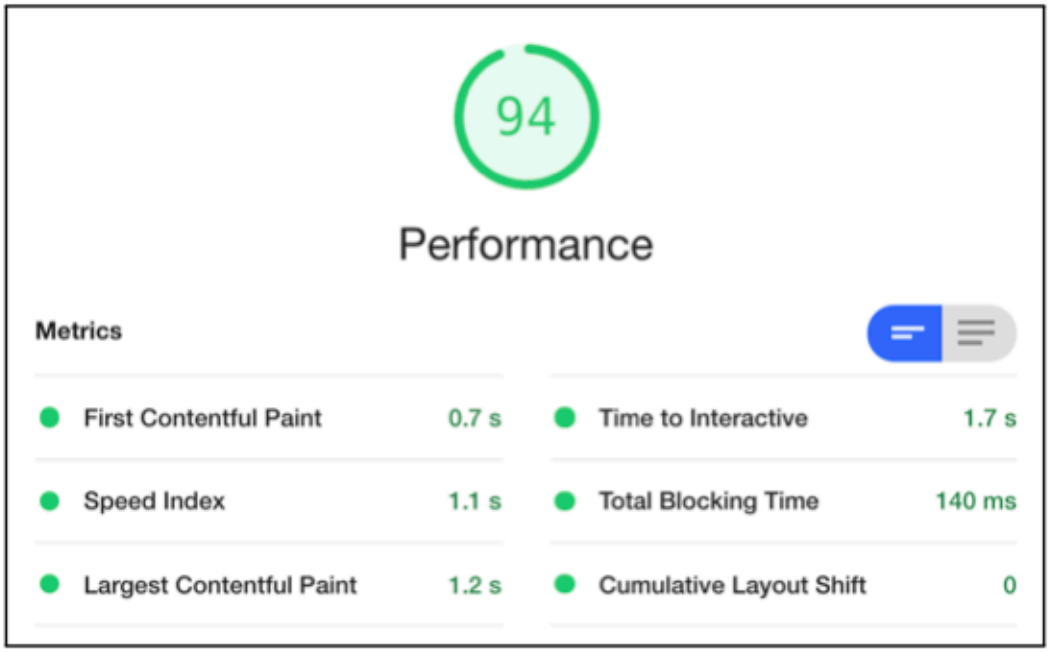

Abbildung 7.1: Lighthouse-Wert der [redacted]-Webseite

	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Result (error code)	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In
First View (Run 1)	0.193s	2.600s	2.614s	2.640s	2.700s	0	2.997s	0.002	4.149s	70	1,281 KB	4.339s	70	1,281

Abbildung 7.2: Webpagetest.org grafische Inhalt-Übersicht

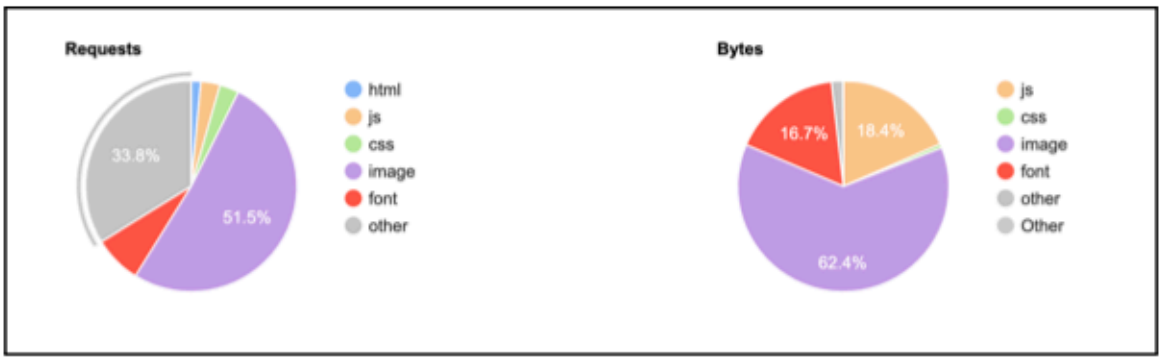

Abbildung 7.3: Webpagetest.org Übersicht – Links: Gesamtanzahl der Anfragen: 70 Rechts: Gesamtanzahl der Bytes: 1281

7.1.3 Kompatibilität

Die Kompatibilität einer Software zu anderen Softwaresystemen wird nach ISO 25010 durch die Faktoren der „Interoperabilität“ und der „Koexistenz“ gemessen. Der Grad der Interoperabilität bemisst sich unter anderem an der Existenz und Qualität einer Softwareschnittstelle zum Austausch von Informationen. Das Maß der Koexistenz beschreibt die Fähigkeit von Systemen, negative Auswirkungen auf die Funktionalität umliegender Systeme zu vermeiden.

Die zur Umsetzung der [redacted]-Webseite benötigten Scrivito-Widgets stellen in sich geschlossene Systeme dar. Ihre Kommunikation über das Scrivito-SDK beschränkt sich auf den Austausch der für sie nötigen Informationen. Sie können somit problemlos unabhängig genutzt oder in andere Scrivito-Widgets integriert werden. Damit erfüllen sie die Anforderungen an Interoperabilität und Koexistenz.

Um Interoperabilität sicherzustellen, bietet das erarbeitete Internationalisierungssystem außerdem eine Schnittstelle, die das Auslesen und Bearbeiten von sprachenabhängigen Inhalten und Konfigurationsparametern ermöglicht. Sie wird durch das Plugin exportiert und kann so nach dessen Einbindung programmtechnisch verwendet werden. Weiterhin bietet die erarbeitete Internationalisierungsfunktion die Möglichkeit, benutzerdefinierte Scrivito-Widgets in das System einzugliedern.

Gleichzeitig befindet sich das Internationalisierungsplugin in einem eigenen JavaScript-Kontext und wird bei der Bedienung innerhalb eines JavaScript-Iframes in den Scrivito-Editor eingebettet. So kann beispielsweise auf Debug-Funktionen, die durch das Internationalisierungsplugin in der Entwicklerkonsole verfügbar gemacht werden, nur im entsprechenden Kontext zugegriffen werden. Dieser Vorgang verhindert eine versehentliche Kommunikation mit koexistierenden Komponenten sowie eine Überschneidung im Namensraum benutzerdefinierter globaler JavaScript-Funktionen. Die Interoperabilität und Koexistenz der Internationalisierungslösung ist also ebenfalls gewährleistet.

7.1.4 Wartbarkeit

Die Wartbarkeit ist nach ISO 25010 durch eine Reihe unterschiedlicher Merkmale charakterisiert. Diese beschreiben die Möglichkeit, ein Softwaresystem nachträglich anpassen zu können. Die sogenannte „Modularität“ ist beispielsweise ein Gradmesser, inwieweit ein Softwaresystem aus Teilen besteht, die zwar ineinander verzahnt sind, aber dennoch mit minimalem Aufwand voneinander gelöst und ausgetauscht werden können. Eine solche Modularität erleichtert im allgemeinen die Wartung der Software. Die Yigu-Seite mit ihren Scrivito-Widgets und das Internationalisierungssystem sind durch die Verwendung von React modular und komponentenbasiert aufgebaut und erfüllen damit im wesentlichen die Anforderungen an die Wartungsfähigkeit. Weiterhin sind komplizierte Funktionen im Quellcode vollständig dokumentiert. Um den Quelltext stilistisch einheitlich zu halten, wurde darüber hinaus eine Abwandlung des „JavaScript Standard Style“ benutzt [2]. Dieser definiert Regeln zum Formatieren von Quelltext.

7.1.5 Benutzerfreundlichkeit und Portabilität

Um die Benutzeroberfläche der Internationalisierungsfunktion so gut wie möglich in Scrivito einzugliedern, wurde sie so angepasst, dass ihre Darstellungselemente denen der Benutzeroberfläche von Scrivito ähnlich sehen. Weiterhin ist die Installation des Internationalisierungsplugins über das JavaScript-Ökosystem so einfach wie möglich gehalten. Zum Einbinden des Internationalisierungsplugins in ein existierendes Scrivito-System muss

es nur über NPM heruntergeladen und anschließend mittels einer Zeile Quelltext eingebunden werden. Da das Internationalisierungsplugin quelloffen auf der „NPM Package Registry“ verfügbar ist, ist die Portabilität ebenfalls gewährleistet.

7.1.6 Sicherheit

Bekannte Schwachstellen aller Abhängigkeiten der entwickelten Lösungen können mit dem NPM Kommando

```
npm audit
```

angezeigt werden. Vor der Behandlung dieser Schwachstellen zeigte die Kommandozeile folgendes Ergebnis:

```
found 5002 vulnerabilities (5001 low, 1 high) in 1987 scanned
   packages
run ‘npm audit fix‘ to fix 5001 of them.
1 vulnerability requires semver-major dependency updates.
```

Die aufgedeckten Schwachstellen konnten mit den `npm` Parameter `audit fix --force` behoben werden.

7.2 Evaluation von Scrivito

In den folgenden Unterkapiteln wird das Content-Management-System Scrivito anhand verschiedener Kriterien kurz untersucht und bewertet. Diese Bewertungen beruhen zum Teil auf persönlichen Maßstäben und Präferenzen. Die Evaluation gibt die Sicht des Verfassers wieder und beinhaltet darüber hinaus programmiertechnische Details.

7.2.1 Benutzbarkeit

Obwohl im allgemeinen ausgereift, finden sich auch in Scrivito einige Schwachstellen. Unter anderem ist die von Scrivito gelieferte Dokumentation in Bezug auf Implementierungsdetails teilweise unübersichtlich. Es existiert keine Gesamtübersicht über alle vorhandenen Methoden der Scrivito-SDK. Zudem liefert die Scrivito-SDK keine Dokumentation über ihre Funktionen im Quellcode, sodass während der Entwicklung eine immer ständige Suche nach Implementierungsdetails in der Online-Dokumentation nötig ist. Die von der Scrivito-SDK gelieferten Funktionen besitzen keine im Quelltext enthaltene Dokumentation und bieten der Programmierumgebung somit keine Möglichkeit, während der Programmierung Hilfestellungen zu geben. Weiterhin sind die Funktionen der Scrivito-SDK naturgemäß nicht quelloffen, was die Fehlersuche mitunter erschwert. Die Online-Dokumentation von Scrivito-Funktionen gibt oft keinen Aufschluß darüber, ob Parameter verpflichtend oder optional sind. Zudem geben einige Hilfestellungen und Artikel der Scrivito-Dokumentation nicht den neuesten Stand der Implementierung wieder. Beispielsweise lieferte das Tutorial zum Hinzufügen von Scrivito in ein React-Projekt

zum Zeitpunkt der Ausarbeitung inkorrekte Informationen zur Einrichtung einer Scrivito-Instanz.

Weiterhin problematisch ist, dass Scrivito innerhalb seines Editors ein anderes CSS-Stylesheets benutzt, als in der Ansicht der Webseite ohne Aufruf durch den Editor. Einige Elemente und Widgets werden beispielsweise um ein Margin oder Padding erweitert, so dass sich die Entwicklung nach konkreten Designvorgaben mühsam gestaltet. Um auf diese Problematik zu reagieren, mussten die CSS-Regeln von Scrivito innerhalb einiger Widgets mit einem sogenannten CSS-Reset zurückgesetzt werden. Scrivito erlaubt es zudem nicht, Komponenten seiner Benutzeroberfläche programmtechnisch zu nutzen. Um die Darstellung der erarbeiteten Internationalisierungslösung dem Aussehen der Scrivito-Benutzeroberfläche anzugleichen, mussten deshalb die entsprechenden Komponenten aufwändig nachgebaut werden.

Positiv ist hingegen, dass der Scrivito-Editor die Möglichkeit bietet, mehrere Arbeitskopien anzulegen und zu verwalten. Zusätzlich können Änderungen unmittelbar im Content-Management-System angezeigt werden. Das erleichtert die Zusammenarbeit mit mehreren Editoren. Zudem erlaubt Scrivito es, Teams und Benutzer des Content-Management-Systems zu verwalten und diesen individuelle Rechte im System zuzuweisen. Damit können mit Hilfe der entwickelten Internationalisierungslösung nicht nur internationale Inhalte unmittelbar im Content-Management-Systems übersetzt werden, sondern auch die übersetzten Inhalte vor der Veröffentlichung überprüft werden. Der komplette Internationalisierungsvorgang des Plugins findet also im Gegensatz zu einem in Kapitel 4 erläuterten herkömmlichen Internationalisierungsvorgang vollständig innerhalb des Systems statt und erfordert keine zusätzliche Software. Weiterhin ist es einem Übersetzer möglich, Fehler noch vor der Veröffentlichung zu entdecken.

7.2.2 Funktionalität

In Bezug auf die Funktionalität weist die Scrivito-SDK einen Schwachpunkte auf. Es ist nicht möglich, eigene Attribute zur Nutzung innerhalb von Widgets oder Objekten zu definieren. Betrachtet man die in 6.5.1 dargestellten möglichen Attribute beim Anlegen einer Widget-Klasse, so fällt auf, dass es kein Attribut gibt, welches genau ein Widget beinhaltet. Es kann nur eine Widgetliste als Attribut angelegt werden. Auf Nachfrage bei den Scrivito-Entwicklern konnte keine Möglichkeit gefunden werden, diese Funktion nachträglich hinzuzufügen. Dies erschwert die Kombination mehrerer Scrivito-Widgets. Ein Editor könnte beispielsweise versehentlich an einer Stelle, an der ein Text-Widget liegt, unerwünscht ein weiteres Widget hinzufügen, was wiederum negative Konsequenzen auf das geplante Layout hat.

8 Fazit und Ausblick

Im folgenden Kapitel werden die in Kapitel 3 analysierten Anforderungen sowie die aus ihnen entstandene Konzeption und der Entwicklungsvorgang des daraus resultierenden Proof-of-Concept zusammengefasst. Der abschließende Ausblick stellt verschiedene Möglichkeiten vor, die in der Thesis erarbeitete Implementierung zu erweitern oder zu verbessern.

8.1 Fazit

Das Ziel dieser Bachelorthesis war die Umsetzung einer internationalen Webseite mit dem Content-Management-System Scrivito. Bei dieser handelte es sich um die Landing-Page des Joint-Venture Unternehmens [redacted]. Um die internationale Komponente der Webseite zu liefern, musste für Scrivito eine Internationalisierungslösung entwickelt werden. Diese sollte über eine grafische Benutzeroberfläche bedient werden können.

Zur Einführung in das Thema wurden im Rahmen der Ausarbeitung zuerst in Kapitel 2 Grundlagen erklärt, die über das vermittelte Wissen eines durchschnittlichen Informatik-Studiums hinausgehen. Diese sind für das allgmeine Verständnis unerlässlich.

Anschließend wurden die nötigen Anforderungen im Kapitel der Analyse erarbeitet. Diese Anforderungen beinhalten Kriterien, welche durch die implementierte Lösung unbedingt erfüllt werden mussten oder optional zu erfüllen waren. Die Analyse in Kapitel 3 lieferte beispielsweise die Erkenntnis, dass die Designvorgaben zur Umsetzung der [redacted]-Webseite in verschiedene Komponenten zerlegt werden mussten. Zusätzlich wurde die Notwendigkeit erkannt, spezielle Werkzeuge zur Unterstützung verschiedener Internet-Browser zu nutzen. Dies gewährleistet die korrekte Funktionsweise einer internationalen Internetseite für Englisch und Chinesisch sprechende Zielgruppen. Weiterhin wurde erkannt, dass die Internationalisierungslösung für Scrivito in der Lage sein sollte, die getrennte Verwaltung mehrsprachiger Haupt- und Unterseiten zu ermöglichen.

Mithilfe dieser Anforderungen wurde in Kapitel 5 die Umsetzung der Problemstellungen konzipiert. Die Designvorgaben der [redacted]-Webseite wurden in Komponenten unterteilt, um die spätere Implementierung so modular wie möglich gestalten zu können. Jeder Komponente wurde ein Konzept zur visuellen Darstellung und Funktionsweise zugeordnet. Damit ist der Leser in der Lage, den gedanklichen Vorgang des Autors dieser Ausarbeitung bei der Implementierung anhand eines Designkonzeptes vollständig

nachzuvollziehen. Weiterhin wurden für die Internationalisierungslösung Grundgerüste angefertigt, welche eine mögliche Darstellung des später für Scrivito implementierten Internationalisierungsplugins aufzeigen.

Anhand dieser Konzepte konnte anschließend eine programmtechnische Implementierung erstellt werden. Im Kapitel der Implementierung konnten anschließend Quellcode-Beispiele dargestellt werden, um dem Leser die Arbeitsweise in der Entwicklung der [redacted]-Webseite und des Internationalisierungsplugins zu erläutern. Er wird an die Entwicklung eines einfachen Scrivito-Widgets und alle dafür nötigen Konzepte herangeführt und ist anschließend ebenfalls in der Lage, die Funktionsweise der Internationalisierungslösung nachzuvollziehen. Zusätzlich wird dem Leser vermittelt, wie das Projekt konkret zu installieren und auszuführen ist.

Die in dieser Ausarbeitung erarbeitete Lösung ist in der Lage, einer mit Scrivito erstellten Webseite Funktionen zur Verwaltung von internationelen Inhalten hinzuzufügen. Sie wurde innerhalb von 3 Monaten kontinuirlich entwickelt. Mit der Verwendung der Internationalisierungslösung ist es möglich, einer existierenden Scrivito-Seite Sprachen zuzuordnen und sie anschließend mit internationalen Inhalten zu füllen. Das Internationalisierungsplugin ermöglicht Ergosign, zum jetzigen Zeitpunkt die projektbasierte Entwicklung internationaler Webseiten mit Scrivito und ist somit von großer wirtschaftlichem Interesse für das Unternehmen.

8.2 Ausblick

Um die Internationalisierungsfunktion zu vervollständigen, muss die Importierungslogik nachträglich hinzugefügt werden. In der Zukunft könnten zusätzlich Arbeitsabläufe entworfen und implementiert werden, welche sich mit der weiteren Verwaltung von mit dem Plugin erstellten Haupt- und Unterseiten beschäftigen. Denkbar wäre eine Funktion, Widgets, die in eine mit dem Internationalisierungsplugin verwaltete Seite eingefügt werden, auch an äquivalente Position anderssprachiger Versionen dieser Seite einzusetzen. Alternativ könnten Widgets auf mehrsprachigen Versionen von Seiten manuell miteinander verknüpft werden.

Ebenfalls vorstellbar ist eine Funktion zum Vergleich von verschiedensprachigen Versionen einer Seite. Da Scrivito bereits Funktionen liefert, Scrivito-Seiten innerhalb eines gesonderten Fensters darzustellen, könnte eine Funktion entworfen werden, die in ihrer Art und Weise dem Kommandozeilenwerkzeug „diff“ gleicht. Damit wäre ein Nutzer des Plugins einfacher in der Lage, Stellen zu erkennen, deren Inhalt noch anzupassen ist.

Es besteht außerdem die Möglichkeit, ein konfigurierbares Menü bereitzustellen, welches die Registrierung von Subdomains und deren Verknüpfung mit den entsprechenden Versionen von Haupt- und Unterseiten ermöglicht. Damit könnten mit dem Plugin erstellte Haupt- und Unterseiten durch ihre Subdomain im Format *de.domain.com* anstatt durch einen Pfadpräfix unterschieden werden.

Der derzeitigen Umsetzung der [redacted]-Webseite fehlen aufgrund der zeitlichen Einschränkung in der Ausarbeitung dieser Thesis einige wenige Implementierungsdetails. Fehler in der Darstellung der [redacted]-Webseite konnten durch einen für Ergosign typischen, zyklischen Prozess aus Design-Review und Nachbesserung weitestgehend ausgemerzt werden. Weiterhin sollte ein Verfahren zur Überprüfung und Fehlerbeseitigung des Quelltextes mit einem Senior-Entwickler vollzogen werden. Um die Ladegeschwindigkeit zu verbessern, könnte die Größe von Bildern und Schriftarten der [redacted]-Seite mit einem Komprimierungswerzeug reduziert werden. Weiterhin ist es möglich, eine zusätzliche Reduzierung von Ladezeit zu gewinnen, in dem Bild- und Videomaterial der Webseite mit Lazy-Loading nur bei Bedarf geladen werden. Diese Verbesserungsvorschläge sollen nach der Beendigung der Thesis und dem Abschluss des Colloquiums in die ausgearbeitete Implementierung eingefügt werden. Die in der Thesis ausgearbeitete Implementierung könnte durch diese Vorschläge noch weiter verbessert werden.

Literatur

[1] *A Brief Introduction to Scrivito*. en. URL: https://www.scrivito.com/a-brief-introduction-to-scrivito-10dd2c5367ac9f12 (besucht am 14. 09. 2020).

[2] Feross Aboukhadijeh. *JavaScript Standard Style*. URL: https://standardjs.com/ (besucht am 29. 09. 2020).

[3] *About Node.Js*. en. URL: https://nodejs.org/en/about/ (besucht am 03. 09. 2020).

[4] Abdulmajeed Alameer und William G. J. Halfond. „An Empirical Study of Internationalization Failures in the Web". In: *International Conference on Software Maintenance*. Jan. 2016, S. 88–98. DOI: 10.1109/ICSME.2016.55.

[5] Tab Atkins-Bittner, Elika J. Etemad und Florian Rivoval. *CSS Snapshot 2018*. Jan. 2019. URL: https://www.w3.org/TR/css-2018/ (besucht am 09. 07. 2020).

[6] Vangie Beal. *Acronym Guide to Web Stacks - Webopedia.Com*. en. URL: https://www.webopedia.com/quick_ref/webstack_acronyms.asp (besucht am 30. 07. 2020).

[7] Goldy Benedict. *Goldy Benedict's Answer to What Is the Difference between a Web Application and a Web Site? - Quora*. Juli 2019. URL: https://www.quora.com/What-is-the-difference-between-a-web-application-and-a-web-site/answer/Goldy-Benedict (besucht am 05. 08. 2020).

[8] *Better for the Editorial Team*. en. URL: https://www.scrivito.com/better-for-the-editorial-team (besucht am 14. 09. 2020).

[9] *Browser Market Share China*. en. Aug. 2020. URL: https://gs.statcounter.com/browser-market-share/all/china/ (besucht am 19. 09. 2020).

[10] *Browser Market Share Worldwide*. en. Juni 2020. URL: https://gs.statcounter.com/browser-market-share (besucht am 29. 07. 2020).

[11] *Can I Use - Arrow Function*. URL: https://caniuse.com/?search=arrow%20functions (besucht am 19. 09. 2020).

[12] *Caveats · Babel*. en. URL: https://babeljs.io/ (besucht am 28. 07. 2020).

[13] Tantek Celik, Elika J. Etemad, Daniel Glazman, Ian Hickson, Peter Linss und John Williams. *Selectors Level 3*. Jan. 2018. URL: https://www.w3.org/TR/selectors-3/ (besucht am 13. 07. 2020).

[14] Chetan Conikee. *An Oxymoron : Static Analysis of a Dynamic Language (Part 3)*. en. Juni 2020. URL: https://securityboulevard.com/2020/06/an-oxymoron-static-analysis-of-a-dynamic-language-part-3/ (besucht am 07. 09. 2020).

[15] *Create React App · Set up a Modern Web App by Running One Command*. en. URL: https://create-react-app.dev/ (besucht am 19. 09. 2020).

[16] *Creating Multi-Language Websites*. en. n.d. URL: https://www.scrivito.com/creating-multi-language-websites-76437a7e2d84fc15 (besucht am 12. 08. 2020).

[17] *Css-Modules/Css-Modules*. css-modules. Sep. 2020. URL: https://github.com/css-modules/css-modules (besucht am 01. 09. 2020).

[18] *Css-Modules/Icss*. css-modules. Juni 2017. URL: https://github.com/css-modules/icss (besucht am 01. 09. 2020).

[19] *Customizing the Routing*. en. URL: https://www.scrivito.com/customizing-the-routing-4ec39c4a63fe5929 (besucht am 07. 09. 2020).

[20] *Cypress - Web Security*. en. URL: https://docs.cypress.io/guides/guides/web-security.html (besucht am 19. 09. 2020).

[21] Mark Davis. *UTS #35: Unicode Locale Data Markup Language*. Apr. 2020. URL: http://www.unicode.org/reports/tr35/tr35-59/tr35.html\#Identifiers (besucht am 14. 09. 2020).

[22] *DefinitelyTyped/DefinitelyTyped*. DefinitelyTyped. Juli 2020. URL: https://github.com/DefinitelyTyped/DefinitelyTyped (besucht am 28. 07. 2020).

[23] Alexis Deveria. *Can I Use... Compare Chrome 84 and QQ Browser 10.4*. URL: https://caniuse.com/\#compare=chrome+84,and_qq+10.4 (besucht am 29. 07. 2020).

[24] Alexis Deveria. *Can I Use... Compare Crhome 84 and UC 12.12*. URL: https://caniuse.com/\#compare=chrome+84,and_uc+12.12 (besucht am 29. 07. 2020).

[25] Alexis Deveria. *Can I Use... Homepage*. URL: https://caniuse.com/ (besucht am 24. 09. 2020).

[26] Alexis Deveria. *Can I Use... Support Tables for HTML5, CSS3, Etc.* URL: https://caniuse.com/ (besucht am 29. 07. 2020).

[27] Houssein Djirdeh. *Reduce JavaScript Payloads with Code Splitting*. Nov. 2018. URL: https://web.dev/reduce-javascript-payloads-with-code-splitting/ (besucht am 07. 08. 2020).

[28] *End to End Testing Framework*. en. URL: https://www.cypress.io/how-it-works (besucht am 19. 09. 2020).

[29] *Enums · TypeScript*. URL: https://www.typescriptlang.org/docs/handbook/enums.html (besucht am 28. 07. 2020).

[30] *Ergosign GmbH Gründet Joint Venture Mit ARTOP Group – UX Und Service Design Für Den Chinesischen Markt*. Juni 2019. URL: https://www.ergosign.de/de/news/2019/news-Ergosign-ARTOP-Joint-Venture.html (besucht am 07. 09. 2020).

[31] Bert Esselink. *The Evolution of Localization*. en. Bd. 4. Language International World Directory. Amsterdam: John Benjamins Publishing Company, Sep. 2000. ISBN: 978-90-272-1955-8 978-1-58811-005-3 978-90-272-1956-5 978-1-58811-006-0 978-90-272-9818-8. DOI: 10.1075/liwd.4. URL: http://www.jbe-platform.com/content/books/9789027298188 (besucht am 10. 07. 2020).

[32] David Flanagan. *JavaScript: The Definitive Guide*. en. Sixth. O'Reilly Media, Inc., Mai 2011. URL: https://www.oreilly.com/library/view/javascript-the-definitive/9781449393854/ (besucht am 13. 07. 2020).

[33] Martin Fowler. *Bliki: TransparentCompilation*. Feb. 2013. URL: https://martinfowler.com/bliki/TransparentCompilation.html (besucht am 28. 07. 2020).

[34] Martin Fowler. *Patterns of Enterprise Application Architecture*. English. 1 edition. Boston: Addison-Wesley Professional, Nov. 2002. ISBN: 978-0-321-12742-6.

[35] Brad Frost. *Atomic Design*. Nov. 2016. URL: https://atomicdesign.bradfrost.com/ (besucht am 25. 08. 2020).

[36] *Getting Started*. en. URL: https://create-react-app.dev/docs/getting-started (besucht am 19. 09. 2020).

[37] Robert Gibb. *What Is a Web Application? | How a Web Application Works*. en. Mai 2016. URL: https://blog.stackpath.com/web-application/ (besucht am 05. 08. 2020).

[38] *Gnu.Org - Gettext*. en. Apr. 2020. URL: https://www.gnu.org/software/gettext/ (besucht am 09. 07. 2020).

[39] Google. *Lighthouse | Tools for Web Developers | Google Developers*. URL: https://developers.google.com/web/tools/lighthouse (besucht am 17. 09. 2020).

[40] *Google/Traceur-Compiler Code Frequency*. en. URL: https://github.com/google/traceur-compiler/commit/caa7b751d5150622e13cdd18865e09681d8c6691 (besucht am 28. 07. 2020).

[41] Infopark Group. *Infopark Group - Powering Digital Transformation*. en. URL: https://infopark.com/en (besucht am 11. 09. 2020).

[42] *HTML & CSS - W3C*. URL: https://www.w3.org/standards/webdesign/htmlcss\#whatcss (besucht am 13. 07. 2020).

[43] *Handbook - Compiler Options*. en. URL: https://www.typescriptlang.org/docs/handbook/compiler-options.html (besucht am 07. 09. 2020).

[44] *Handbook - The TypeScript Handbook*. en. URL: https://www.typescriptlang.org/docs/handbook/intro.html (besucht am 01. 09. 2020).

[45] „ECMAScript® 2020 Language Specification". en. In: (Juni 2020). Hrsg. von Jordan Harband und Kevin Smith, S. 860.

[46] Phil Hawksworth. *What Is a Static Site Generator? How Do I Find the Best One to Use?* en. Apr. 2020. URL: https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/ (besucht am 05. 08. 2020).

[47] *I18n for WordPress Developers « WordPress Codex*. URL: https://codex.wordpress.org/I18n_for_WordPress_Developers (besucht am 23. 09. 2020).

[48] *ISO - ISO 3166 — Country Codes*. en. Nov. 2013. URL: https://www.iso.org/iso-3166-country-codes.html (besucht am 17. 09. 2020).

[49] *ISO - ISO 639 — Language Codes*. en. Feb. 2003. URL: https://www.iso.org/iso-639-language-codes.html (besucht am 17. 09. 2020).

[50] *ISO 25010*. März 2011. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (besucht am 17. 09. 2020).

[51] *Introducing Hooks – React*. en. URL: https://reactjs.org/docs/hooks-intro.html (besucht am 13. 08. 2020).

[52] *Introduction to the Server Side*. en. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction (besucht am 03. 08. 2020).

[53] Alex Ivanovs und ContributorWriter. *WYSIWYG Website Builders for Online Business*. en. 19:38:24 -0500. URL: https://www.huffpost.com/entry/wysiwyg-website-builders_b_8814882 (besucht am 14. 09. 2020).

[54] *JavaScript End to End Testing Framework | Cypress.Io*. URL: https://www.cypress.io/ (besucht am 19. 09. 2020).

[55] David Karlton und Karlton. *Naming Things Is Hard | Dk*. en-US. Blog. Dez. 2017. URL: https://www.karlton.org/2017/12/naming-things-hard/ (besucht am 11. 09. 2020).

[56] Maxim Koretskiy. *Maximkoretskiy/Postcss-Autoreset*. Juli 2020. URL: https://github.com/maximkoretskiy/postcss-autoreset (besucht am 29. 07. 2020).

[57] Julian Krieger. *WIP: Add Typings for the Public SDK of Scrivito CMS by Juliankrieger · Pull Request #46834 · DefinitelyTyped/DefinitelyTyped*. en. URL: https://github.com/DefinitelyTyped/DefinitelyTyped/pull/46834 (besucht am 10. 09. 2020).

[58] Luis A. Leiva und Vicent Alabau. „Automatic Internationalization for Just In Time Localization of Web-Based User Interfaces“. In: *acm transactions on computer human interaction* 22.3 (Mai 2015). DOI: 10.1145/2701422.

[59] Håkon Wium Lie. *Cascading HTML Style Sheets – A Proposal*. Okt. 1994. URL: https://www.w3.org/People/howcome/p/cascade.html (besucht am 09. 07. 2020).

[60] Håkon Wium Lie. „PhD Thesis: Cascading Style Sheets“. Diss. University of Oslo, März 2005. URL: https://www.wiumlie.no/2006/phd/\#h-28 (besucht am 21. 07. 2020).

[61] *Loaders*. en. URL: https://webpack.js.org/loaders/ (besucht am 19. 09. 2020).

[62] John W. Long. *Sass vs. SCSS: Which Syntax Is Better?* Sep. 2011. URL: http://thesassway.com/editorial/sass-vs-scss-which-syntax-is-better (besucht am 07. 09. 2020).

[63] Alexander Madyankin, zhouwenbin, Dan Freeman und Semirulnik. *Css-Modules/Postcss-Modules*. css-modules. Sep. 2020. URL: https://github.com/css-modules/postcss-modules (besucht am 01. 09. 2020).

[64] Ryan Seddon Published: March 21st und 2012 Comments: 5 Your browser may not support the functionality in this article. *Introduction to JavaScript Source Maps - HTML5 Rocks*. en. URL: https://www.html5rocks.com/en/tutorials/developertools/sourcemaps/ (besucht am 28. 07. 2020).

[65] Brody McKee. *Mrmckeb/Typescript-Plugin-Css-Modules*. Aug. 2020. URL: https://github.com/mrmckeb/typescript-plugin-css-modules (besucht am 21. 08. 2020).

[66] Steven Mendelez. *The Difference Between Dynamic & Static Web Pages | Chron.Com*. Aug. 2018. URL: https://web.archive.org/web/20190320233700/https://smallbusiness.chron.com/difference-between-dynamic-static-pages-69951.html (besucht am 05. 08. 2020).

[67] Jason Miller und Addy Osmani. *Rendering on the Web*. en. 2, 2019. URL: https://developers.google.com/web/updates/2019/02/rendering-on-the-web (besucht am 07. 08. 2020).

[68] *NPM.Js - Scrivito*. URL: https://www.npmjs.com/package/scrivito (besucht am 10. 09. 2020).

[69] *Naming Convention / Methodology / BEM*. URL: /methodology/naming-convention/ (besucht am 01. 09. 2020).

[70] *Netlify: All-in-One Platform for Automating Modern Web Projects*. URL: https://www.netlify.com/ (besucht am 05. 08. 2020).

[71] *Node.Js*. en. URL: https://nodejs.org/en/ (besucht am 03. 09. 2020).

[72] Node.js. *Overview of Blocking vs Non-Blocking*. en. URL: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/ (besucht am 11. 09. 2020).

[73] *Npm | Build Amazing Things*. URL: https://www.npmjs.com/ (besucht am 13. 08. 2020).

[74] *Open Brand*. Mai 2007. URL: https://www.opengroup.org/openbrand/register/brand3555.htm (besucht am 13. 08. 2020).

[75] *Plugins · Babel*. en. URL: https://babeljs.io/docs/en/plugins/ (besucht am 28. 07. 2020).

[76] *Post-Processing CSS*. en. URL: https://create-react-app.dev/docs/post-processing-css (besucht am 19. 09. 2020).

[77] *ReactDOM – React - Hydrate*. en. URL: https://reactjs.org/docs/react-dom.html\#hydrate (besucht am 14. 09. 2020).

[78] *Responsive Web Design - What It Is And How To Use It*. en. Jan. 2011. URL: https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/ (besucht am 24. 09. 2020).

[79] Nicolò Ribaudo. *7.8.0 Released: ECMAScript 2020, .Mjs Configuration Files and @babel/Cli Improvements · Babel*. en. Jan. 2020. URL: https://babeljs.io/blog/2020/01/11/7.8.0 (besucht am 28. 07. 2020).

[80] Ben Rogojan. *JAMStack vs MEAN vs LAMP | Web Stack Guide | ButterCMS | ButterCMS*. URL: https://buttercms.com/blog/jamstack-vs-mean-vs-lamp-your-guide-to-picking-one (besucht am 30. 07. 2020).

[81] Margaret Rouse. *What Is a Content Management System (CMS)? Definition from WhatIs.Com*. en. URL: https://searchcontentmanagement.techtarget.com/definition/content-management-system-CMS (besucht am 06. 09. 2020).

[82] *Sass - Syntactically Awesome Stylesheets*. URL: https://web.archive.org/web/20130901145805/http://sass-lang.com/about.html (besucht am 25. 07. 2020).

[83] *Sass: Sass Basics*. URL: https://sass-lang.com/guide (besucht am 25. 07. 2020).

[84] *Sass: Syntax*. URL: https://sass-lang.com/documentation/syntax (besucht am 07. 09. 2020).

[85] Michael Satran. *Understanding Internationalization - Win32 Apps*. en-us. Mai 2018. URL: https://docs.microsoft.com/en-us/windows/win32/intl/understanding-internationalization (besucht am 10. 07. 2020).

[86] *Scrivito CMS - Better for Enterprise*. en. URL: https://www.scrivito.com/product (besucht am 11. 09. 2020).

[87] *Scrivito CMS - Top-Level API*. en. URL: https://www.scrivito.com/scrivito-sdk (besucht am 11. 09. 2020).

[88] Scrivito. *How Does a Scrivito App Work?* en. URL: https://www.scrivito.com/how-does-a-scrivito-app-work-68623f3ceeff1c5d (besucht am 14. 09. 2020).

[89] Scrivito. *Jamstack for Web Projects*. Techn. Ber. 2019. URL: https://www.scrivito.com/JAMstack-for-web-projects.

[90] Scrivito. *What Is New in the Scrivito Example App?* en. URL: https://www.scrivito.com/getting-productive-with-our-example-app-in-2020-e80ea274f88e2f97 (besucht am 14. 09. 2020).

[91] *See Who Is Using Babel*. URL: https://babeljs.io/ (besucht am 28. 07. 2020).

[92] Ben Shapiro. *Website vs. Web Application: What's the Difference?* en-US. Apr. 2013. URL: https://www.seguetech.com/website-vs-web-application-whats-the-difference/ (besucht am 05. 08. 2020).

[93] Andrey Sitnik. *Postcss/Autoprefixer*. PostCSS. Juli 2020. URL: https://github.com/postcss/autoprefixer (besucht am 29. 07. 2020).

[94] Andrey Sitnik. *Postcss/Postcss*. PostCSS. Juli 2020. URL: https://github.com/postcss/postcss (besucht am 29. 07. 2020).

[95] *Solution Stack*. URL: https://encyclopedia2.thefreedictionary.com/Solution+stack (besucht am 09. 07. 2020).

[96] Soma Somasegar. *TypeScript: JavaScript Development at Application Scale*. en-us. Jan. 2012. URL: https://docs.microsoft.com/en-us/archive/blogs/somasegar/typescript-javascript-development-at-application-scale (besucht am 28. 07. 2020).

[97] *Stack Overflow Developer Survey 2020*. URL: https://insights.stackoverflow.com/survey/2020/?utm_source=social-share\&utm_medium=social\&utm_campaign=dev-survey-2020 (besucht am 13. 07. 2020).

[98] *Tablet Screen Resolution Stats Worldwide*. en. Aug. 2020. URL: https://gs.statcounter.com/screen-resolution-stats/tablet/worldwide (besucht am 23. 09. 2020).

[99] Shannon Tan. *Translators Go beyond Language as Web Sites Cross Borders*. Aug. 2001. URL: https://www.webcitation.org/5uabN4muG?url=http://articles.chicagotribune.com/2001-08-06/business/0108060190_1_localization-translation-web-sites (besucht am 09. 07. 2020).

[100] *Template View [Web Application Component Toolkit]*. Dez. 2012. URL: https://web.archive.org/web/20121204081204/http://www.phpwact.org/pattern/template_view (besucht am 31. 07. 2020).

[101] Tex Texin. *Origin Of The Abbreviation I18n For Internationalization*. URL: http://www.i18nguy.com/origini18n.html (besucht am 09. 07. 2020).

[102] *The State of Babel · Babel*. en. URL: https://babeljs.io/blog/2016/12/07/the-state-of-babel (besucht am 28. 07. 2020).

[103] *Tutorial: Intro to React – React*. en. URL: https://reactjs.org/tutorial/tutorial.html (besucht am 08. 08. 2020).

[104] *Typed JavaScript at Any Scale*. en. URL: https://www.typescriptlang.org/ (besucht am 01. 09. 2020).

[105] *Using Navigations and the Hierarchy Browser*. en. URL: https://www.scrivito.com/using-navigations-and-the-hierarchy-browser-02653972c9ebbce0 (besucht am 24. 09. 2020).

[106] *V8 JavaScript Engine*. URL: https://v8.dev/ (besucht am 03. 09. 2020).

[107] Juan Vega. *Client-Side vs. Server-Side Rendering: Why It's Not All Black and White*. en. Feb. 2017. URL: https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/ (besucht am 07. 08. 2020).

[108] *Visual Studio Code - Code Editing. Redefined*. en. URL: https://code.visualstudio.com/ (besucht am 13. 08. 2020).

[109] *W3Techs - Web Surveys*. URL: https://w3techs.com/ (besucht am 09. 07. 2020).

[110] WHATWG. *HTML Standard*. URL: https://html.spec.whatwg.org/multipage/dom.html\#classes (besucht am 01. 09. 2020).

[111] *WebPageTest - Website Performance and Optimization Test*. URL: https://webpagetest.org/ (besucht am 17. 09. 2020).

[112] *Webpack-Contrib/Css-Loader*. webpack-contrib. Aug. 2020. URL: https://github.com/webpack-contrib/css-loader (besucht am 01. 09. 2020).

[113] *Webpack*. en. URL: https://webpack.js.org/ (besucht am 19. 09. 2020).

[114] *What Is a CDN? | How Do CDNs Work?* en-us. URL: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (besucht am 14. 09. 2020).

[115] *What Is a Landing Page? Landing Pages Explained*. en-US. URL: https://unbounce.com/landing-page-articles/what-is-a-landing-page/ (besucht am 24. 09. 2020).

[116] *What Is a Polyfill?* URL: https://remysharp.com/2010/10/08/what-is-a-polyfill/ (besucht am 29. 07. 2020).

[117] Allen Wirfs-Brock und Brendan Eich. „JavaScript: The First 20 Years". In: *Proceedings of the ACM on Programming Languages* 4.HOPL (Juni 2020), 77:1–77:189. DOI: 10.1145/3386327. URL: https://doi.org/10.1145/3386327 (besucht am 09. 07. 2020).

[118] Mohammad Younes. *MohammadYounes/Rtlcss*. Juli 2020. URL: https://github.com/MohammadYounes/rtlcss (besucht am 29. 07. 2020).

[119] Jake Zatecky. *Jakezatecky/React-Checkbox-Tree*. Sep. 2020. URL: https://github.com/jakezatecky/react-checkbox-tree (besucht am 10. 09. 2020).

[120] *@babel/Core*. URL: https://www.npmjs.com/package/@babel/core (besucht am 28. 07. 2020).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

CSS „Cascading Style Sheets“

HTML „Hypertext Markup Language“

SASS „Syntactically Awesome Style Sheets“

DRY „Don’t repeat yourself“

MDN „Mozilla Developer Network“

CMS „Content-Management-System“

CMA „Content-Management-Application“

CDA „Content-Delivery-Application“

SCSS „Sassy CSS“

WYSIWYG „What-You-See-Is-What-You-Get“

CDN „Content-Delivery-Network “

POT „Portable-Objects-Template“

PO „Portable-Object“

MO „Machine-Object“

Anhang

A Anhang

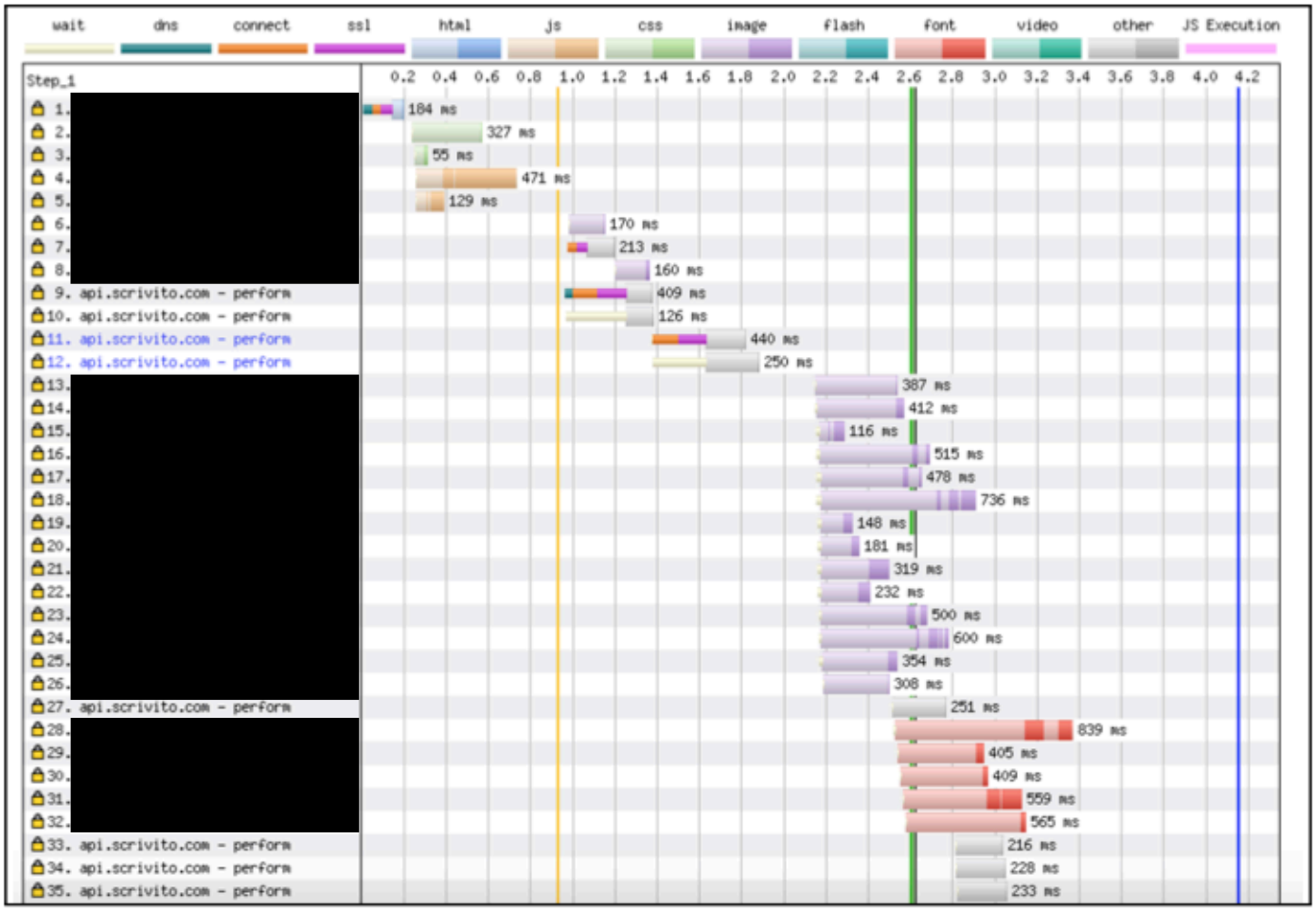

Abbildung A.1: Webpagetest.org Ergebnis Teil 1

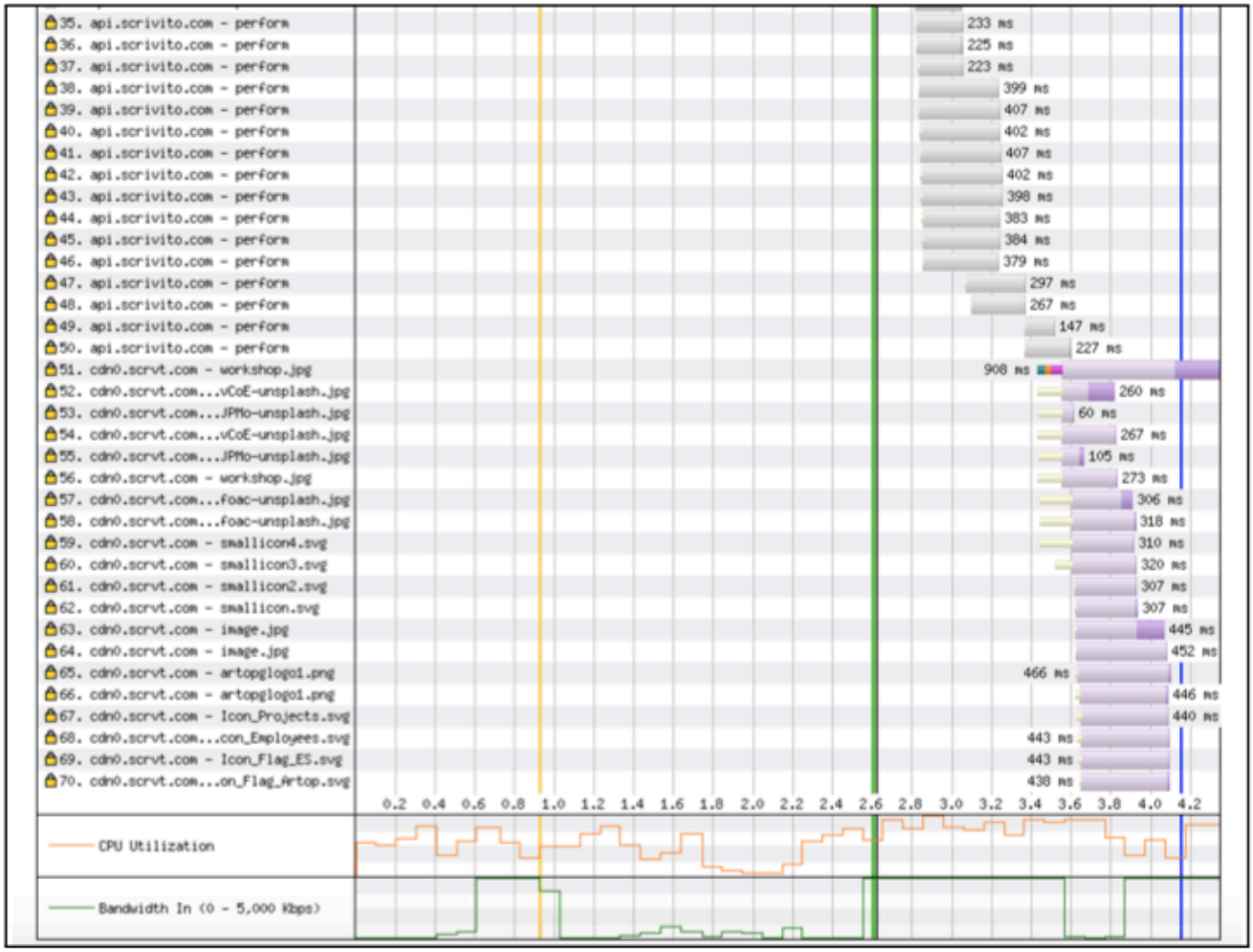

Abbildung A.2: Webpagetest.org Ergebnis Teil 2

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt