HOCHSCHULE
MITTWEIDA
University of Applied Sciences

Fakultät Angewandte Computer- und Biowissenschaften

Professur Digitale Transformation und Angewandte Medieninformatik

Bachelorarbeit

Vergleich ausgewählter JavaScript Front-End-Frameworks am Beispiel eines webbasierten Budgetierungstools

Vinzenz Herrmann

Mittweida, den 3. November 2022

Erstprüfer: Prof. Dr.-Ing. Christian Roschke

Zweitprüfer: M. Sc. Dominik Breck

Herrmann, Vinzenz
Vergleich ausgewählter JavaScript Front-End-Frameworks am Beispiel eines webbasierten Budgetierungstools
Bachelorarbeit, Fakultät Angewandte Computer- und Biowissenschaften
Hochschule Mittweida — University of Applied Sciences, November 2022

Name: Herrmann, Vinzenz
Studiengang: Medieninformatik und Interaktives Entertainment
Seminargruppe: MI17w1-B
English Title: Comparison of selected JavaScript front-end frameworks using the example of a web-based budgeting tool

Inhaltsverzeichnis

1. Einführung und Motivation

Dieses Kapitel dient als Einstieg in die vorliegende Arbeit und beschäftigt sich mit den fundamentalen Aspekten wie der Aufgabenstellung und Motivation, auf denen diese Arbeit basieren soll. Weiter wird der generellen Aufbau der Arbeit sowie deren Zielstellung betrachtet.

1.1. Motivation

Durch vorangegangene Auseinandersetzung mit der Entwicklung von Webanwendungen des Authors, unter anderem im Zuge eines Praktikums, entwickelte sich ein zunehmendes Interesse an der Thematik. Da eine Vielzahl von Möglichkeiten besteht derartige Anwendungen umzusetzen, soll diese Arbeit einen Einblick in die Entwicklung von Webapplikationen anhand von drei ausgewählten JavaScript Front-End-Frameworks liefern und vertiefen. Dabei soll ein möglichst praxisnaher Vergleich anhand eines konkretisierten Fallbeispiels eben dieser Frameworks durchgeführt werden, um die Entscheidungsfindung, welche Technologie genutzt werden soll, zu vereinfachen. Hier soll diese Arbeit ansetzen und Anhand des praktischen Beispiels eines Budgetierungstools dessen Front-End Komponente betrachten und mittels der JavaScript Frameworks umsetzen und dokumentieren.

1.2. Aufgabenstellung

Die Ausführung dieser Arbeit setzt sich mit der Konzeption, Entwicklung und Evaluation der Front-End Komponente eines webbasierten Budgetierungstools mit Hilfe der JavaScipt-Frameworks React, Svelte und Vue auseinander. Dabei ist die Kernaufgabe der Wissenschaftlichen Arbeit die Frage zu klären, welches der gennanten

Frameworks, anhand definierter Kriterien, sich für die Umsetzung des Budgetierungstools am besten eignet.

1.3. Zielstellung und Aufbau der Arbeit

Ziel dieser Arbeit ist es an ein bereits bestehendes System, welches zur Budgetierung von Forschung- und Entwicklungsprojekten an deutschen Hochschulen dient, anzuschließen und dieses zu erweitern. Dieses Tool wurde mittels des Open-Source Content Management Systems OctoberCMS realisiert. Dabei soll an das bestehende System angeknüpft werden und die Front-End Komponente betrachtet werden. Da es viele verschiedene Möglichkeiten gibt, das Front-End von Softwareanwendungen zu realisieren, sei es allein die Entscheidung von serverseitigem oder clientseitigem rendern, oder auch die Auswahl der Technologien. Diese Arbeit setzt sich mit drei verschiedenen JavaScript Front-End Frameworks auseinander, um Unterschiede und Gemeinsamkeiten sowie Performancedaten zu analysieren und ein Fazit hervorzubringen, welches der zu prüfenden Frameworks sich für das konkrete Fallbeispiel am besten eignet.

Der Aufbau dieser Wissenschaftlichen Arbeit gliedert sich in sechs Kapitel, wobei diese sich mit folgenden Prioritäten befassen. Das erste Kapitel beschäftigt sich mit der Motivation der Arbeit, sowie der Einführung in die Problemstellung. Aufbauend darauf erfolgt die Erläuterung des gesetzten Ziels der Arbeit und deren Struktur. Innerhalb des zweiten Kapitels liegt der Fokus auf der Vermittlung aller theoretischen Grundlagen. Im dritten Kapitel erfolgt zunächst eine Analyse der bestehenden Problemstellung, dabei sollen Konzepte für die Anforderungen, sowie für die Implementation und den Vergleich erstellt werden. Anschließend sollen die für Testzwecke erstellten Implementationsschritte im vierten Kapitel einer genaueren Erläuterung durchlaufen. Diese Umsetzungen liefern die Grundvoraussetzung für das fünfte Kapitel, in welchem dann anhand der Analyse und des Vergleichkonzeptes die Umsetzungen evaluiert werden sollen. Abschließend wird im sechsten Kapitel eine Zusammenfassung durchgeführt, sowie einen Ausblick für potentielle weitere Forschungen bezüglich dieser Problemstellung in Aussicht gestellt.

2. Grundlagen

Dieses Kapitel befasst sich mit den Grundlagen von webbasierten Anwendungen. Dabei werden diese definiert, sowie auf für diese Arbeit relevante Technologien und Architekturen eingegangen, um einen Überblick für alle weiteren Kapitel zu erzeugen. Des weiteren bieten die hier vermittelten Grundlagen die Voraussetzung für den praktischen Teil dieser Arbeit.

2.1. Webbasierte Anwendungen

Als Webanwendung wird eine Anwendung verstanden, welche auf dem Client-Server Prinzip basiert und auf Webtechnogologien wie unter anderem HTTP, HTML, PHP, CSS und JavaScript setzt. Meist erfolgt die Nutzung und der Aufruf einer solchen Applikation über einen Webbrowser als Benutzerschnittstelle, wie Firefox oder Chrome. Dabei wird die Anwendung serverseitig bereitgestellt und anschließend in ihren Bestandteilen vom Browser interpretiert und angezeigt. Dabei stellt der Server die Laufzeitumgebung für die Anwendung dar, die sich dann über Hintergrundsysteme erstreckt und auf diese zugreifen kann, welche Datenverarbeitung und Persistenz, sowie Datenbankeninteraktion beinhalten können.[Roh18, S. 1 ff.]

Anders als bei nativen Anwendungen ist keine Installation einer Webapplikation notwendig, da diese vom Webbrowser ausgeführt wird. Eine weitere Eigenschaft von Webanwendungen spiegelt sich in ihrer plattformunabhängigen Verwendbarkeit wider, wodurch eine Anwendung reichen kann um alle Plattformen abzudecken. [Dig21]

2.1.1. Client Server Architektur

Die Client-Server-Architektur oder auch Client-Server-System besteht aus einem oder mehreren Clienten die Funktionen und Daten des Servers anfordern, sowie einem Server der diese bereitstellt. So bilden beide Bestandteile in der Summe ein System, mit verschiedenen Zuständigkeiten. Ein Server kann in diesem System mehrere Clienten, wie in Abbildung 2.1 durch mehrere Verbindungen zwischen Clienten (links) und Server (mitte) dargestellt, parallel bedienen, wobei die Clienten selbst in keinem Verhältnis zueinander stehen. Physisch betrachtet können Client und Server durch verschiedene Rechensysteme repräsentiert werden, aber auch auf einem einzigen Rechner laufen. Die Interaktion zwischen beiden folgt einem festen Muster aus Anfrage und Antwort.[Ben14, S. 21 ff.] Unter dem Client versteht man ein Programm, welches Verbindungen aufbauen kann, und anschließend requests sendet. Bei dem Client kann es sich beispielsweise um einen Webbrowser handeln, er kann jedoch auch durch eine andere Programmart verkörpert werden. Unter dem Server versteht man das Programm welches Verbindungen empfängt, dabei Anfragen interpretiert und versteht, und anschließend durch das Rücksenden von Antworten bedient. [Wil99, S. 60]

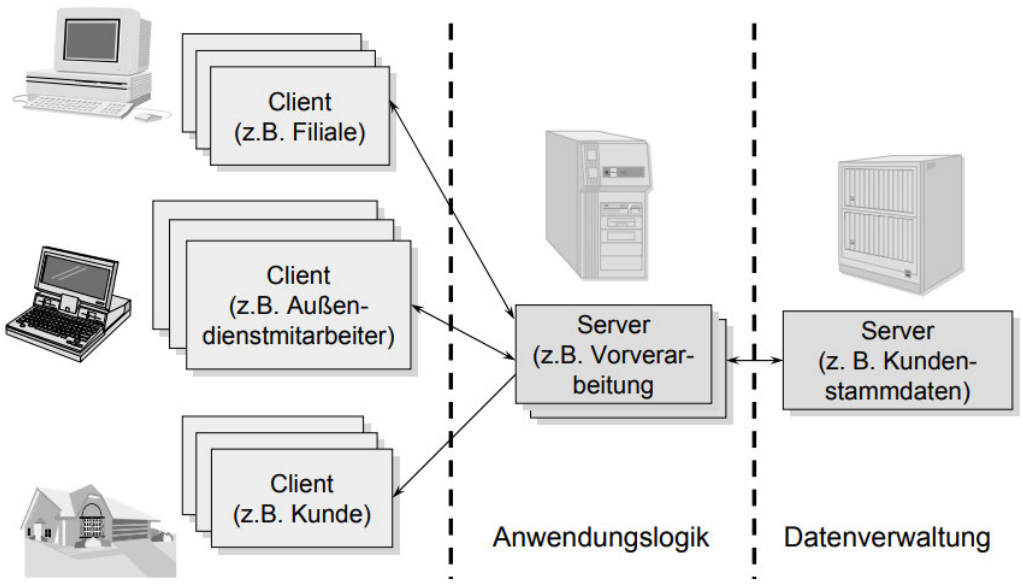

Abbildung 2.1.: Client/Server Modell [Sch12, S. 23]

Eine Client-Server-Architektur mit mindestens drei Untergliederungen haben sich im Laufe der Webentwicklung durchgesetzt. Dabei klassifizieren sich diese, wie in Abbildung 2.1 zu sehen, in die clientseitige Präsentationsschicht, sowie den serverseitigen Teil, repräsentiert durch die Anwendungslogik, welche die Verarbeitung von Daten und Prozessen durchführt und in den persistenten Teil, der Datenverwaltung. Diese Untergliederung liefert die Möglichkeit Schnittstellen zwischen den einzelnen Stufen zu schaffen, was Umgestaltungen begünstigt, da auf diese Art die Stufen unabhängig voneinander austauschbar sind.[Sch12, S. 23 f.]

2.1.2. Hypertext Markup Language

Hypertext Markup Language oder abgekürzt HTML ist das Fundament einer Website. Über sie ergibt sich der strukturelle Aufbau, sowie der Inhalt. Das HTML-Markup besteht aus Tags wodurch Elemente und deren Inhalt für die Anzeige im Webbrowser spezifiziert werden können. [MDN21] Die Strukturierung des HTML-Dokuments ergibt sich dabei aus dem Document Object Model (DOM). Das DOM bestehend aus Knoten, welche einem Objektbaum zugeordnet sind, geben Aufschluss über den Aufbau der Website und die Verbindung zu der Knoten untereinander. Der Knoten enthält dabei Referenzen zum jeweiligen Eltern- oder Kindknoten. Zudem repräsentiert er ein HTML-Element mit seinen Eigenschaften, Inhalt und Events. [LBR+19, S. 12]

2.1.3. Cascading Stylesheet

Das Cascading Stylesheet (CSS) wird zur Deklaration des optischen Erscheinungsbildes einer Website bzw. Webanwendung genutzt. Über CSS lässt sich das Layout als auch Schriftarten, Elementpositionen, Farben als auch Animationen definieren.[Att20, S. 1]

Für die vereinfachte Erstellung von Layouts mittels CSS stehen verschiedene Frameworks, wie das von Twitter entwickelte CSS Framework ”Bootstrap” zur Verfügung. Die Vorteile eines solchen besteht darin, dass eine Vielzahl von vorgefertigten Templates, Themen und Bausteinen für die Gestaltung einer Website bereitgestellt werden. Dabei bietet Bootstrap sowohl Lösungen für die gestalterische Optimierung einer Seite für Mobile Endgeräte als auch für die Desktopansicht. [Kra20, S. 1]

2.1.4. Hypertext Transfer Protocol

HTTP kurz für Hypertext Transfer Protocol ist ein zustandsloses Protokoll zur Übertragung von Daten und Webseiten zwischen Server und Client. Dieses arbeitet mit response/request also einem Antwort/Frage Prinzip, wobei eine Antwort nur serverseitig getätigt werden kann. [Smi15, S. 101]

HTTP Request Eine HTTP Request wird vom Client an den Server geschickt, diese Anfrage hat einen bestimmten Aufbau, welcher sich aus Kopfzeile, Anfrageparameter und Nutzdatenteil zusammensetzt. Teil dieser Anfrage ist die HTTP-Methode welche in der Kopfzeile wiederzufinden ist. Diese spezifiziert die durchzuführende Aktion des Servers. Einige beutende HTTP-Methoden sind wie in Tabelle 2.1 dargestellt: GET, POST, PUT und DELETE. [Abt19, S. 157 ff.]

HTTP Methode	**Funktion**
GET	Zum anfordern von Ressourcen vom Server.
POST	Zum Übertragen von Daten an den Server.
PUT	Zum Erstellen oder Verändern von Ressourcen auf dem Server.
DELETE	Zum Löschen von Ressourcen auf dem Server.

Tabelle 2.1.: HTTP Methoden [Abt19, S. 157 ff.]

Der Nutzdatenteil enthält beispielsweise Daten welche mittels der Methode POST an den Server übermittelt werden sollen. [Abt19, S. 157 ff.]

HTTP Response Die HTTP-Antwort setzt sich ebenfalls aus Kopfzeile, Anfrageparameter und Nutzdatenteil zusammen, jedoch unterscheiden sich die Inhalte. Die Kopfzeile enthält die Protokoll Version und den Status Code, wobei dieser die Werte aus Tabelle 2.2 annehmen kann. Dabei teilt der Statuscode dem Clienten mit wie die Anfrage ausgeführt wurde. Der Nutzdatenteil enthält die vom Client angeforderte Ressource. [Abt19, S. 160 f.]

HTTP Status Code	Bedeutung
100 - 199	Informative Meldung.
200 - 299	Erfolgreiche Anfrage.
300 - 399	Weitergeleitete Anfrage.
400 - 499	Fehlerhafte Anfrage.
500 - 599	Server Fehler.

Tabelle 2.2.: HTTP Status Codes [Abt19, S. 160 f.]

2.2. Entwurfsmuster

Um bestimmte Eigenschaften wie die Verständlichkeit und Erweiterbarkeit von Softwaresystem zu verbessern ist der Einsatz von bewährten Mustern in der Softwareentwicklung etabliert. Für Problemstellungen, die mit der Entwicklung von Systemen einhergeht, liefern solche Muster Lösungsansätze, welche sich bereits vorher bewährt haben. Dabei arbeiten Entwurfsmuster oft mit Abstraktionen und sind als Blaupause für den Aufbau einer Anwendung zu verstehen. [Gol14, S. 64 ff.]

2.2.1. Model View ViewModel

Das Entwurfsmuster "Model View ViewModel" (MVVM) untergliedert sich in drei verschiedene Bestandteile. Diese drei Komponenten ergeben sich aus dem Namen dieses Musters. Dabei repräsentiert das Model die Logik und das Datenmodell. Die View stellt die Benutzer Schnittstelle dar. Das ViewModel vertritt den gesamten Zustand und die Verhaltensweise der View. Der wesentliche Unterschied zwischen der View und dem ViewModel besteht darin, dass die View für die reine visuelle Respräsentation des ViewModels veranwortlich ist und keine weitere Logik, außer die Anbindung an das ViewModel enthalten darf. Ereignisse die in der Oberfläche stattfinden müssen an das ViewModel weitergeleitet werden, daraufhin erfolgt eine Aktualisierung des ViewModel. Durch die Datenbindung zwischen View und ViewModel folgt anschließend bei Änderungen von Zuständen im ViewModel die anpassung der View. [JDF+14, S. 162 f.]

2.2.2. Model View Controller

Das Muster ”Model View Controller” (MVC) gliedert ein System in drei Komponenten. Diese Bestandteile stellen das Model, welches für die Verabeitungslogik und das Datenmodell steht, die View repräsentativ für die Ausgabe, und den Controller, welcher für die Eingabe verantwortlich ist, dar. Bei Anwendung dieses Musters ist es möglich die unterschiedlichen Funktionen eines Programms in getrennten Bestandteilen zu implementieren. Dabei soll die Datenhaltung, die Präsentationslogik und die Interpretation von Nutzereingaben nicht vermischt werden. [Gol14, S. 377 f.]

2.3. JavaScript

Urpsrünglich wurde JavaScript unter einem anderen Namen veröffentlicht. So wurde das heutige JavaScript anfangs unter den Namen LiveScript und Mocha im Jahr 1995 für den Netscape Navigator Browser von Brendan Eich entworfen. Durch eine Kooperation von Netscape und Sun, dem Unternehmen hinter der Programmiersprache Java, erfolgte aus Marketing-Gründen eine Umbenennung in JavaScript. [Ack21, S. 45]

Nach der Erscheinung von JavaScript entwickelte Microsoft eine ähnliche Sprache für den Internet Explorer. Um eine Standard des Sprachkerns von JavaScript zu erzielen, reichte Netscape als Folge dessen JavaScript bei der European Computer Manufacturers Association (ECMA) ein. Daraufhin erfolgte die Standardisierung unter dem Namen ECMAScript (ES). Durch die jährliche Erweiterung dieses Standards erfolgte die Versionierung seit der in 2015 veröffentlichten Version 6 unter einem Kürzel, welches sich aus der Abkürzung von ECMAScript und dem Kalenderjahr der Veröffentlichung zusammensetzt. So wird ECMAScript seit Version 6 als ES2015 abgekürzt [Ack21, S. 45]

In der Web-Entwicklung kann JavaScript für eine Vielzahl von Anwendungsfällen zum Einsatz kommen. Zu diesen zählt unter anderem die Integration von Interaktivität und einem dynamischen Verhalten einer Website. Unter diesen Funktionalitäten fallen beispielsweise Validierungen von Formularen, sowie Sortierungen, Filterungen, sowie ein- und ausblenden bestimmter Seitenelemente. Auch das dynamische Laden von Seiteninhalten über AJAX und deren Injektion an gebenener Stelle im HTML-Dokument ist ein weiteres Anwendungsfeld.[Ack21, S. 65 f.]

JavaScript ist die einzige Programmiersprache welche vom Browser interpretiert werden kann. Dabei dient sie beispielsweise zur Erstellung von browserspezfischem CSS-Styling und HTML-Inhalten, sowie spezieller audiovisueller Effekte. Mit JavaScript lassen sich Front-End Webapplikationen mit lokaler Datenspeicherung, aber auch eine komplette verteilte Webapplikation mit remote Datenspeicherung implementieren, wobei die Remote-Datenspeicherung durch eine traditionelle serverseitige Sprache wie PHP, aber auch durch JavaScript und NodeJS umgesetzt werden kann. [GW17, S. 10]

JavaScript gehört zu den prototypbasierten Sprachen, das heißt es handelt sich um eine objektorientierte aber klassenlose Sprache. Objekte können dabei durch das Kopieren bereits existenter Objekte, den Prototypen, erzeugt werden, wobei alle Eigenschaften des bereits existierenden Prototyps übernommen werden können. Dabei sind diese veränder- und ergänzbar. [MDN20]

2.3.1. Variablen und Datentypen

JavaScript ist eine typenlose Sprache, dass heißt das Variablen Deklarationen zwar einen Namen festgelegt bekommen können, jedoch ohne einen zugewiesenen Datentypen. Daher bekommt die Variable erst einen Datentypen zugewiesen, sobald ihr ein Wert, welcher einem Typ entspricht, zugewiesen wird. Die Tabelle 2.3 liefert dabei eine Übersicht über die verschiedenen Datentypen die eine JavaScript Variable annehmen kann. [Ste14, S. 54]

Zur Verwendung von Variablen muss zunächst die Variablendeklaration erfolgen. Diese setzt sich aus dem Schlüsselwort var und dem Variablennamen zusammen. Seit ES2015 ist es auch möglich das Schlüsselwort let zur Variablendeklaration verwenden. Variablen lassen sich jederzeit überschreiben, soll dies vermieden werden, muss eine Konstante mit dem Schlüsselwort const deklariert und ihr Wert initialisiert werden. [Ack21, S. 87]

Typ	Beschreibung
Number	Kann Ganzzahl- aber auch Gleitkommawerte enthalten
String	Zeichenkette welche diverse alphanumerische Zeichen enthalten kann.
Boolean	Wahrheitswert welcher true oder false annehmen kann
Object	Datentyp zum speichern von Objektreferenzen
null	Sonderdatentyp für den Fall kein Wert bei einer Objektreferenz vorliegt
undefined	Sonderdatentyp, eine Variable besitzt diesen Wert bis ihr ein anderer Wert zugewiesen wird.

Tabelle 2.3.: JavaScript Datentypen [Ste14, S. 56]

2.3.2. Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (Ajax), beschreibt eine Webtechnik, welche es ermöglicht dynamisch einzelne Inhalte einer Website nachzuladen. Dabei erfolgt die Client - Server Kommunikation über eine HTTP-Anfrage, welche vom Server ausgewertet und anschließend mit einer HTTP-Antwort erwidert wird. Anders als bei der synchronen Kommunikation müssen Fixbereiche wie beispielsweise die Navigationsleiste und Bestandteile dessen Inhalt nicht von der Anfrage betroffen sind, nicht als komplett neu generierte Seite an den Client gesendet werden. [Ack21, S. 545 ff.]

Eine Grundlage für Ajax-Anfragen stellte das JavaScript-Objekt XMLHttpRequest dar, wobei eine neuere Alternative zum XMLHttpRequest-Objekt, die FetchAPI, asynchrone Anfragen zu versenden, vereinfacht. Die JavaScript-Funktion fetch() ist dabei die Grundlage für die Abstraktion aller Typen von asynchronen Anfragen. Als Parameter bekommt diese Funktion dabei die URL und wenn nötig weitere Konfigurationen als zweiten Parameter in Form eines Objektes übergeben. Über dieses Objekt lassen sich unter anderem die HTTP-Methode, die HTTP-Header und

der HTTP-Body konfigurieren. Der Rückgabewert dieser Funktion ist ein Promise-Objekt, auf welches sich verschiedene Methoden anwenden lassen um Fehler abzufangen und das Ergebnis weiter zu verarbeiten.[Ack21, S. 584 ff.]

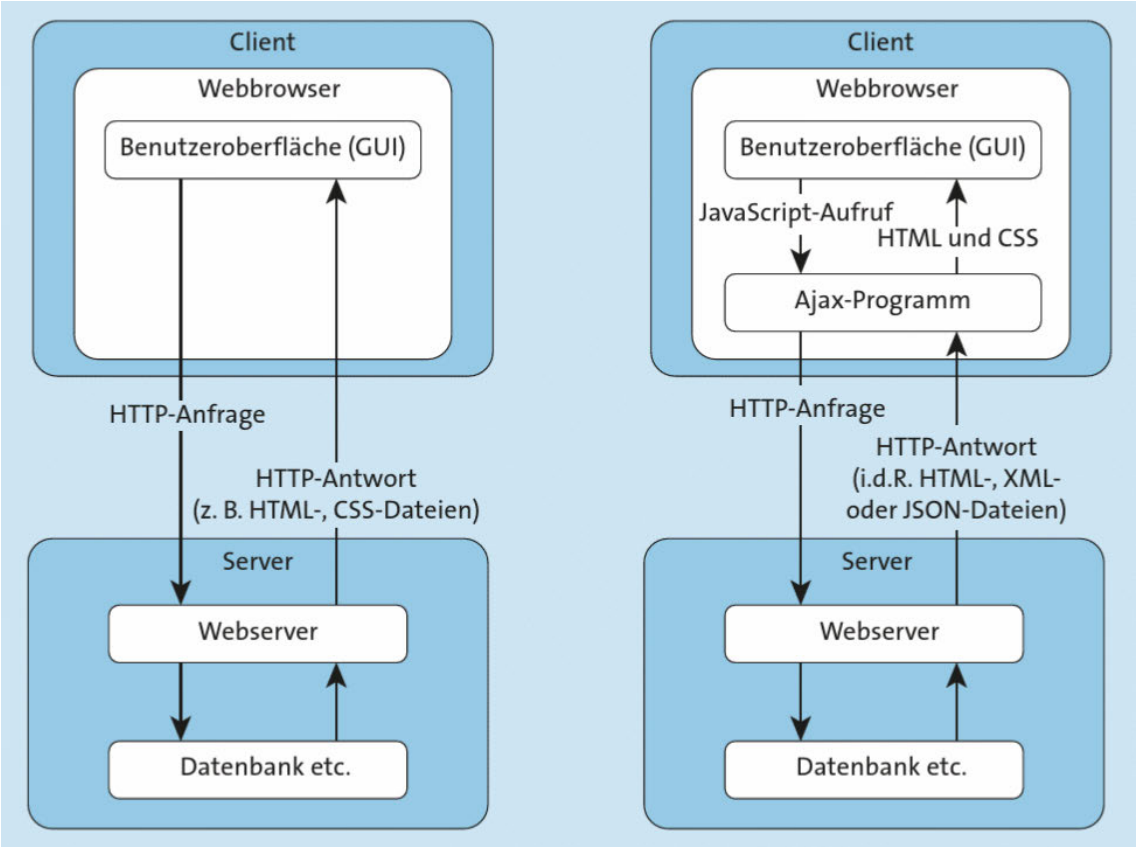

Abbildung 2.2.: Unterschied zwischen AJAX-basierten und klassischen Webanwendungen [Ack21, S. 548]

Die Asynchronität wird dadurch ausgezeichnet, dass die Website weiter arbeitet, während sie eine Anfrage an den Server sendet. Dort wird diese Anfrage verarbeitet und anschließend als Antwort an den Browser zurückgegeben. So können bestimmte Abschnitte, wie in Abbildung 2.2 zu sehen, der Website mit den XML-Daten aufgefrischt werden, anstatt als komplett neue Seite gesendet zu werden. Da auf diese Weise die Website nicht neu übertragen werden muss ergeben sich Vorteile gegenüber Anwendungen die Ajax nicht nutzen. Diese äußern sich in der Möglichkeit, Daten dynamisch zu aktualisieren und validieren, aber auch in kürzeren Antwortzeiten und einer geringeren Severlast, sowie eine verminderte Bandbreitennutzung der Weban-

wendung. Weiter erfolgt eine klare Trennung von Daten, Layout und Format.[Vor08, S. 1]

2.3.3. JavaScript Object Notation

JSON oder auch JavaScript Object Notation ist ein Datenformat welches stark an die Objektnotation JavaScripts angelehnt ist. Syntaktisch besteht eine JSON Datei aus einer Sammlung von Schlüssel/Wert Paaren. [Smi15, S. 37 f.] Dabei können die Werte mit verschiedenen Datentypen beschrieben werden. Diese möglichen Datentypen sind: Strings, Numbers, Objects, Arrays, Boolean und null. [Smi15, S. 42]

2.3.4. Node Package Manager

NPM oder Node Package Manager ist eine Paketverwaltung für JavaScript Apllikationen. Basierend auf der Node.js Laufzeitumgebung mit deren sich JavaScript Quelltext auch außerhalb des Browsers kompilieren lässt. Hauptsächlich kommt NPM zur Installierung und Deinstallierung verschiedener Pakete für JavaScript Anwendungen, sowie zum Verwalten und installieren von Abhängigkeiten und Versionsverwaltung zum Einsatz. [NPM]

2.4. JavaScript Frameworks

Um wiederkehrende Probleme in der Programmierung zu lösen, gibt es Frameworks, diese ermöglichen es durch Bereitstellung von Funktionssammlungen, auf bereits bestehende Problemlösungen zurückzugreifen, anstatt diese selbst zu implementieren. Daher kann ein Framework als Programmiergerüst betrachtet werden, welches sich dadurch auszeichnet spezifische Funktionalitäten zur Verfügung zu stellen. Dabei stellt es einen Rahmen bereit, mit welchem sich eine Anwendung erstellen lässt. In diesem Framework sind bereits Codestrukturen vorhanden, welche sich auf eigene Problemstellungen, anwenden lassen. [Ste18, S. 2]

JavaScript Frameworks dienen dazu verschiedene Probleme in der Webentwicklung möglichst effizient zu lösen. Unter anderem geschieht dies durch die Möglichkeit eine Wiederverwendbarkeit des Quellcodes in verschiedenen Projekten zu ermöglichen. Fälschlicher Weise werden Frameworks und Bibliotheken häufig als das gleiche verstanden, dabei teilen sie Gemeinsamkeiten aber unterscheiden sich dennoch in den Details. Beide bestehen aus bereits vorher geschriebenem Code welcher unter anderem aus Funktionen besteht welche Lösungen für spezifische Probleme zur Verfügung stellen. Bei Bibliotheken lassen sich diese nach Bedarf in das Projekt importieren. Frameworks sind restriktiver, da sie eine Struktur vorgeben um anwendungsspezifische Fälle zu lösen. Eine weiteres wichtiges Unterscheidungsmerkmal ist die ”Inversion of Control”, welche beschreibt wie der Code ausgeführt wird. Dabei wird beispielsweise ein Methodenaufruf einer Bibliothek vom eigenen Quellcode kontrolliert, während bei einem Framework, dieses die Kontrolle über die Ausführung des eigenen Quellcodes hat. [Ran21]

2.4.1. React

React ist eine JavaScript-Bibliothek entwickelt von Meta, ehemals Facebook. React stellt eine Grundstruktur der Ausgabe von Benutzeroberflächenkomponenten einer Website zur Verfügung. Der Hierarchische Aufbau sowie die Implementierung über einen eigenen HTML-Tag sind dabei Eigenschaften dieser Komponenten. Die Veränderungen, die dabei bei einer Website einhergehen finden nicht durch eine direkte Manipulation des DOMs des Webbrowsers statt, sondern erfolgt im Speicher des von React erzeugten virtuellen DOM. Daher können Veränderungen zunächst dort stattfinden, bevor die Modifikation im Browser DOM stattfindet. Somit bietet diese

Bibliothek eine Basis für SPAs (Single-Page-Applikationen). Da Webanwendungen standardmäßig aus mehreren, miteinander verlinkten, HTML-Dokumenten bestehen, wird bei einem Wechsel der Seite eine erneute Anforderung des Clienten an den Server gestellt um diese Seite zu laden. Bei SPAs wird ein einziges HTML-Dokument angefordert, wo durch die dazugehörigen JavaScript Datei, weitere Seiteninhalte, ohne einen neuen Seitenaufruf, nachladen kann. Daher wird die Anwendung einmalig vom Server gesendet und vom Browser empfangen. React dient daher zur Erstellung interaktiver und sich veränderten Benutzeroberflächen. [Wol21, S. 991 f.]

React Komponenten React setzte bereits von vornherein auf Komponentenorientierung, wobei deren Aufbau sich mittlerweile in zwei Möglichkeiten untergliedert, die klassenbasierten Komponenten und die funktionalen Komponenten. Die Verwendung dieser bietet annähernd gleiche Optionen, jedoch liegt der Augenmerk zunehmend auf dem funktionalen Ansatz.[Rod20] Wie in Listing 2.1 zu sehen, stellt diese Komponente eine JavaScript-Funktion dar, welche eine valide React-Komponente repräsentiert, da sie ein einziges Objekt von Eigenschaften als Übergabeparameter zulässt und als Rückgabewert ein React-Element liefert.[reac]

```
1 function Welcome(props) {
2     return <h1>Hallo {props.name}</h1>;
3 }
```

Listing 2.1: React Funktionskomponente [reac]

Alternativ kann eine React-Komponente auch über eine ES6-Klasse, wie in Listing 2.2 demonstriert, definiert werden, jedoch sind aus der Perspektive von React beide Komponenten identisch.[reac]

```
1 class Welcome extends React.Component {
2     render() {
3         return <h1>Hallo {this.props.name}</h1>;
4     }
5 }
```

Listing 2.2: React Klassenkomponente [reac]

Die Syntax für React Komponenten wird JavaScript XML (JSX) genannt. Dabei stellt es eine Mischung aus HTML Tags und Benutzerdefinierten Tags sowie JavaScript, welche für eine React Komponente stehen dar. HTML Markup wird dabei für die für die Definition des Aufbaus und der Bestandteile der Benutzeroberfläche verwendet. [AB20, S. 11]

React Hooks React Hooks beschreiben eine API welche es ermöglicht die Komponenten mit der Funktionalität von React zu verbinden. Einige wichtige dieser Hooks sind useState, useEffect und useContext. Die ”useState” Hook wird verwendet um Komponenten mit einem zustandsbehafteten Wert zu versehen. Wird die Komponente erstmals gerendert kann die useState Hook verwendet werden um diesem Wert einen initial Wert zu versehen. [AB20, S. 64 f.]

```
1 const [name, setName] = useState('Adam');
2 const [age, setAge] = useState(35);
```

Listing 2.3: React State [AB20, S. 64 f.]

Wie im obigen Code Beispiel Listing 2.3 dargestellt, geschieht dies im Funktionsaufruf ”useState”, dabei wird der Variablen ”name” ein String mit dem initial Wert ”Adam” übergeben. Um diesen Wert nun zu verändern gibt die Hook einen Array zurück. Der erste Wert des Arrays stellt den zustandsbehaftet Wert dar, während der zweite Wert eine Funktion zur Veränderung des ersten Wertes ist. Findet nun in der Programmlogik eine Veränderung des zustandsbehafteten Wertes statt, wird die Komponente neu gerendert und der neue Wert dargestellt. [AB20, S. 64 f.]

Die Hook ”useEffect” wird verwendet um Nebeneffekte in einer Komponente auszuführen. Dabei wird dieser eine Funktion übergeben welche beim laden sowie jedem Update der Komponente ausgeführt wird. [AB20, S. 73 f.]

Die ”useContext” Hook dient zum teilen von Daten zwischen verschiedenen Komponenten auf globaler Ebene. Dabei kann aus verschiedenen Komponenten auf den erzeugten Kontext zugegriffen werden. In Fällen wie beispielsweise Informationen zu einem aktuell eingeloggten Nutzer, die Relevant für mehrere Komponenten sein können. [AB20, S. 73 f.]

Event Handling Das Eventhandling in React-Elementen zum entgegenehmen von Benutzerinteraktionen mit der Anwendung ähnelt stark dem des reinen JavaScript in Zusammenhang mit DOM-Elementen. Syntaktisch und Funktional gibt es jedoch kleine Unterschiede. React Events werden statt in Kleinbuchstaben mittels Camel Case bennant, weiter wird dem Eventhandler zum auslösen eines Events kein String, sondern eine Funktion übergeben.[reab]

Bedingtes Rendern Um bestimmte Komponententeile für die Darstellung an eine Bedingung zu knüpfen stehen in React die standartmäßigen JavaScript Operatoren wie die *if*-Anweisung oder der tenärer Operator zur Verfügung.[reaa]

Routing Routing ermöglicht es die Struktur und Ansicht der Anwendung zu verändern. Dabei wird beim URL Routing die vom Browser aktuelle URL verwendet um den Inhalt, der dem Nutzer präsentiert wird anzupassen. Dafür werden Navigationselemente benötigt, die die URL des Browsers verändern, ohne eine dabei eine erneute HTTP-Anfrage auszulösen. Dabei kann eine bestimmte Komponente an den URL Pfad gebunden werden, diese Komponente wird dann geladen sobald diese URL durch das dazugehörige Navigationselement aufgerufen wird. [Fre19, S. 591 ff.]

Für die Verwendung des React-Router muss das Zusatzpaket ”react-router-dom” installiert und anschließend in die betreffende Komponente importiert werden. [Fre19, S. 592 ff.]

2.4.2. Svelte

Svelte wurde 2016 von Rich Harris entwickelt, mit dem Ziel näher an klassischem JavaScript angelehnt zu sein und weniger komponentenspezifischen Code zu schreiben. Dabei kompiliert Svelte sein Markup, ohne die Notwendigkeit, dass das Framework zur Laufzeit arbeitet, in wiederverwertbare JavaScript Module. Auf diese Art müssen keine Bibliotheken während der Laufzeit geladen werden, da Svelte auf einem Compiler beruht, der den Code in ein schmales und optimiertes Bundle verarbeitet. Durch diese schmaleren Bundles wird eine schnelle Ladezeit ermöglicht. Ähnlich wie bei dem Framework React 2.4.1 muss keine direkte Manipulation des DOM durchgeführt werden, jedoch übernimmt Svelte dieses Prozess und beruht anders als React nicht auf einem virtuellen DOM. Ein großer Unterschied besteht darin, dass

der Svelte Komponenten Markup sich mittels HTML Tags gliedert, bestehend aus CSS, HTML und JavaScript Bereichen.[Seg20, S. 22 f.]

Bei der Entwicklung von Webapplikationen mit dem Framwork Svelte wird zusätzlich die Verwendung eines Bundlers benötigt. Dieser liefert unter anderem Features wie die Entfernung ungenutzten Quelltextes sowie Abhängigkeiten. Der Bundler sammelt den gesammten JavaScript Quellcode in eine einzige Datei und compiliert die Komponenten von Svelte in ausführbaren Code.[Seg20, S. 36]

Svelte Komponenten Jede Svelte Komponente besteht aus einer Datei mit der Endung ".svelte". Dabei kann diese Datei jeweils nur eine Komponente enthalten. Anhand des Beispiels aus Listing 2.4 lässt sich der Aufbau einer solchen und deren Untergliederung erkennen. Der Auszeichnende Teil ist definiert mittels HTML Tags wie hier beispielsweise der Text "My first Svelte app". CSS Styling kann in den "style" Tags definiert werden, dieses Styling beschränkt sich allerdings auf diese Komponente. Anschließend kann der JavaScript Code in den ßcriptTTags geschrieben werden.[Seg20, S. 40]

```
1     <h1>Hello, {name}!</h1>
2     <p>My first Svelte app</p>
3     <style>
4         p {
5             color: #1d4585;
6         }
7     </style>
8     <script>
9         export let name = ''
10    </script>
```

Listing 2.4: Svelte Komponente [Seg20, S. 39]

Arten der Datenbindung Die Änderung von zustandsbehaftet Werten also state, bedarf keiner Verwaltungsdaten anders als bei React und Vue, da ein Großteil der Verarbeitung bereits bei der Kompilierung durchgeführt wird und so die Veränderungen direkt in der DOM und nicht in einer virtuellen DOM stattfindet.[Lib22, S. 9 f.] Dabei können Werte an die Eigenschaften von Komponenten gebunden werden wobei diese Datenbindung in zwei Richtungen erfolgen kann, was bedeutet, ändert sich der Wert im Programm, so ändert sich auch die Darstellung in der Benutzeroberfläche, sollte eine Veränderung eines Wertes in der Benutzeroberfläche stattfinden, so ändert sich dieser auch im Programm.[Seg20, S. 71]

Eventhandling Zur Ereignissteuerung stellt Svelte über die Direktive *on:* verschiedene Kontroll Instanzen zur Verfügung, unter anderem können Klicks, aber auch alle anderen Ereignistypen von JavaScript.

```
1    <button on:click="{e => window.alert('message from the click
     handler')}">Click me</button>
```

Listing 2.5: Svelte Eventhandling [Lib22, S. 107]

Dieses Codebeispiel aus Listing 2.5 verdeutlicht wie die Direktive in Svelte anzuwenden ist. Bei einem Klick auf den Erzeugten Button wird die inline Funktion ausgeführt welche ein Fenster im Browser mit einer Meldung erzeugt. [Lib22, S. 107]

Bedingtes Rendern Um über einen Array zu iterrieren und einen Bereich für jedes Element dieses Arrays zu rendern stellt Svelte die {*#each*} Direktive bereit.[Seg20, S. 65]

Svelte Store Um zustandsbehaftete Werte über mehrere Komponenten zugreifbar zu gestalten verwendet Svelte Stores, diese haben eine Erreichbarkeit für jede Komponente in einem Projekt beziehungsweise Anwendung, und enthalten lesbare sowie beschreibbare Werte. Dabei wird eine komplette Reaktivität gewährleistet, das bedeutet sollte eine Komponente von einem Wert des Stores abhängig sein, so wird bei einer Änderung des Wertes eine Neuausführung aller reaktiven Bestandteile, die von diesem Wert abhängig sind, vollzogen.[Seg20, S. 72]

Routing Die Funktionalität des Routings erfolgt analog zu der Beschreibung bei React im Abschnitt 2.4.1. Für die Implementation eines Routers muss bei Svelte ebenfalls auf ein Zusatzpaket, welches den Namen ”svelt-spa-router” trägt zurückgegriffen werden. [Seg20, S. 99]

2.4.3. Vue

Vue ist ein JavaScript Framework welches sich hauptsächlich auf das Entwurfsmuster ”Model View ViewModel” konzentriert. Konzipiert wurde es für Single-Page-Anwendungen, kann jedoch auch für Komplexere Projekte genutzt werden. Bei Vue handelt es sich um ein Open-Source-Framework, welches unter der MIT Lizenz läuft. [Dei22, S. 2 f.]

Eine Vue Anwendung beruht auf der Erstellung einer neuen Vue-Instanz. Dafür wird ein JavaScript Objekt von Optionen dem Konstruktor übergeben. Daher entwickelt sich die komplette Anwendung aus einer Vue-Wurzelinstanz, die wahlweise eine Struktur aus in sich verschachtelten, wiederverwendbaren Komponenten einschließt. Diese Struktur weißt Ähnlichkeiten zum DOM-Baum auf. Nach der Erstellung einer solchen Instanz werden alle Eigenschaften zum reaktiven System von Vue hinzugefügt. Ändern sich dann die Werte dieser Eigenschaften im Datenmodell, wird die Ansicht neu gerendert und damit angepasst um diese Werte auch darzustellen.[Ste19, S. 46]

Vue Komponenten Um eine Komponente in Vue zu erzeugen muss zunächst eine dafür dedizierte ”.vue” Datei angelegt werden. Dabei gliedert sich diese in einen JavaScript und HTML Teil. Im JavaScript Teil der Komponente kann diese als einzige Komponente durch den default export, oder mehrere Komponenten durch einen benannten export, exportiert werden. [vuea]

Arten der Datenbindung Um Elemente mit der View zu Verbinden, gibt es die Direktive v-bind, damit können ein oder mehrere Attribute oder Komponenteneigenschaft dynamisch an einen Ausdruck gebunden werden.[Ste19, S. 50]. Die Datenbindung verlangt diese Direktive jedoch nicht zwangsläufig, über die Moustache-Syntax erfolgt die Bindung automatisch, jedoch kann diese nicht in HTML Attributen verwendet werden.[Ste19, S. 50 f.]

Bedingtes Rendern Analog zur if-Anweisung stellt Vue die v-if Direktive zur Verfügung, um bedingtes Rendern zu ermöglichen. Wird diese Direktive in einem Tag verwendet wird dieser Block nur gerendert wenn ein Wahrheitswert vom Bedingungsausdruck zurückgegeben wird. Weiterhin ist es möglich einen Alternativen Block auszugeben mittels v-else, dafür muss diese Alternative im Quelltext unmittelbar auf das v-if Element folgen.[Ste19, S. 109 ff.]

Eine weitere Option für das bedingte Rendern stellt die *v-for* Direktive dar. Mit dieser lassen sich Arrays analog zur *forEach* Methode von JavaScript durchlaufen, denn anders als bei purem JavaScript gibt die *v-for* Direktive bei jedem Durchlauf das Element zurück, während die *for*-Schleife den Index zurückgeben würde.[Ste19, S. 75 f.]

Eventhandling Für das Eventhandling stellte Vue die Direktive v-on zur Verfügung, welche JavaScript ausführt, wenn sie ausgelöst wird. Alternativ kann auch der Token @ als kürzere Form verwendet werden, so kann beispielweise v-on:click= aber auch @click zur Ereignissteuerung bei einem Klick verwendet werden.[Ste19, S. 121]

Vuex Um Daten zwischen Komponenten auszutauschen stellt Vue Vuex als Lösung zur Verfügung. Mit Vuex können mehrere Komponenten auf einen Datensatz zugreifen. Dadurch wird die Reaktivität außerhalb der Komponentengrenzen verwaltet. Um Vuex in der Applikation verwenden zu können muss zunächst ein Store erstellt werden. Wie in der Abbildung 2.3 zu sehen ist, übernimmt Vuex dabei das gesammte Datenmanagement, sowie die Logik dahinter. Methoden, welche unter anderem zu Veränderung von Daten im Vuex Store füren werden daher nur in den den Komponenten aufgerufen.[Rib20, 327 f.]

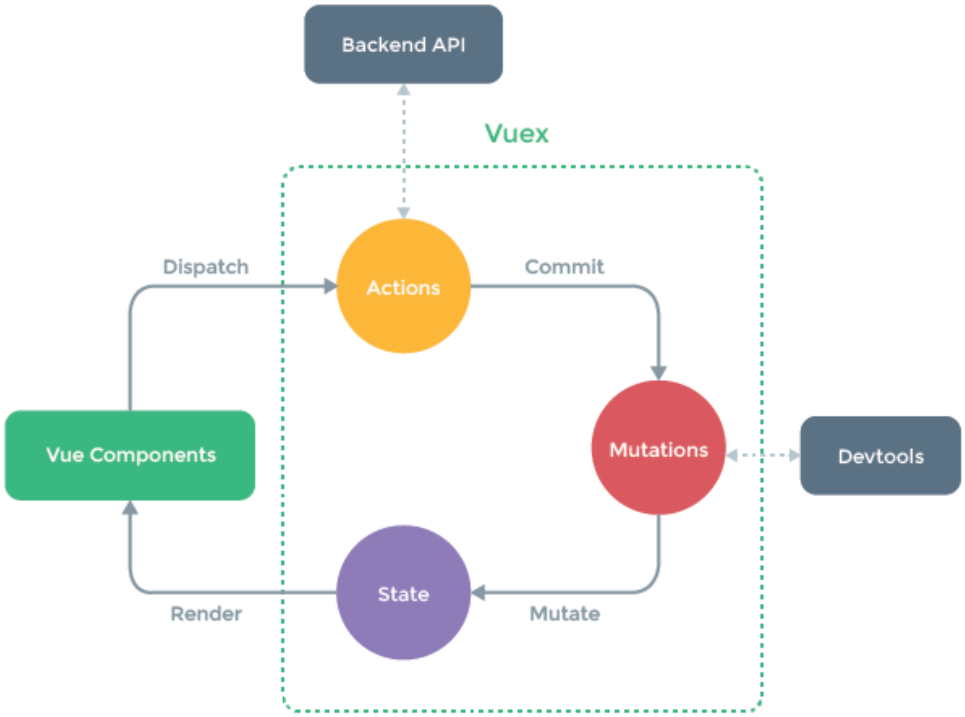

Abbildung 2.3.: Vuex [vueb]

Für die Implementation des Vuex Stores in einem Vue Projekt ist die Zusatzinstallation dieser Bibliothek von Nöten. Dies kann mittels des Paketmanager NPM und dessen Befehl *npm install vuex@next* realisiert werden.[Rib20, S. 20]

Routing Die Funktionalität des Routings erfolgt analog zu der Beschreibung bei React im Abschnitt 2.4.1. Für die Implementation des Routings in einem Vue-Projekt steht die offiziell Unterstützte "vue-router" Bibliothek zur Verfügung. [Ste19, S. 71]

2.5. October CMS

October ist ein Content Management System, bassierenden auf dem PHP-Framework Laravel. Es wurde mit dem Ziel entwickelt, den Entwicklungsprozess in der Webentwicklung einfacher zu gestalten. Dabei ist dieses System Open Source, jedoch nicht kostenlos.[octb] October verspricht eine einfache Skalierbarkeit von Projekten die mit eben diesem CMS erstellt wurde, unter anderem durch die Möglichkeit Plugins einzubinden, welche Interfaces zur Bearbeitung im back-end zur Verfügung stellen. Außerdem soll die Bedienbarkeit über den Verzicht auf zwingende Programmierkenntnisse zur Verwaltung von Daten vereinfacht sein.[octa]

2.5.1. Content Management System

Ein Content Management System, oder kurz CMS, steht für ein System, welches zum bereitstellen, sowie verwalten von Inhalten dient. Dabei steht der zur Verfügung gestellte Inhalt im Vordergrund, während Struktur und technische Basis dem hinten angestellt sind. Bei diesen Systemen können ein oder mehrere Nutzer die Inhalte verwalten, pflegen und das System bearbeiten. Meist folgt dies einem Rollensystem um festlegen zu können, welche Benutzer zu welchen Aufgaben autorisiert sind.[Ste16, S. 4 ff.]

2.6. Web Vitals

In diesem Abschnitt werden verschiedene Messkriterien erläutert, anhand deren die Performance einer Webanwendung gemessen werden kann. Die Web Vitals, welche von Google initiert wurden, sollen dabei eine einheitliche Richtlinie liefern mit welcher sich die Qualitätsmerkmale, sowie die Benutzererfahrung einer Website beziehungsweise Webanwendung bestimmen lassen. [LD22, S. 157]

2.6.1. Core Web Vitals

Die ”Core Web Vitals” stellen dabei eine Untergruppe der ”Web Vitals” dar. Dabei fokussieren sich die ”Core Web Vitals” auf drei Hauptmerkmale. Diese Untergliedern

sich in die Dauer, die es benötigt die Seite zu laden, wie stabil sich die Seite verhält während sie lädt und während eine Nutzer Interaktion stattfindet und abschließend ab wann man mit der Seite interagieren kann. [LD22, S. 157]

Largest Contentful Paint Der Largest Contentful Paint (LCP) stellt ein Maß für die Ladezeit einer Website dar. Ausschlaggebend ist dabei die benötigte Zeit für das Laden des mehrheitlichen Seiteninhalts von dem Zeitpunkt ab wo auf die Seite navigiert wird. Für den Nutzer stellt dies den Zeitpunkt dar, wo er die Seite als fertig geladen wahrnimmt. [LD22, S. 157]

First Input Delay Der First Input Delay (FID) gibt Auskunft über die Dauer die es benötigt bis die geladene Seite und all deren Elemente, von dem Zeitpunkt ab wann auf die Seite navigiert wurde, vollständig Interaktiv nutzbar sind. [LD22, S. 157]

Cumulative Layout Shift Der Cumulative Layout Shift (CLS) gibt Auskunft über die Stabilität einer Seite, in dem gemessen wird, wie sehr sich diese beim laden der Seitenelemente im Browser verändert. Je geringer diese Veränderung auftreten, desto besser fällt die Wertung aus. [LD22, S. 158]

2.6.2. Chrome Lighthouse

Das Werkzeug "Lighthouse", welches in den Google Chrome Webbrowser integriert ist, stellt eine Reihe verschiedener Analysemöglichkeiten für eine Website zur Verfügung. Unter anderem lassen sich Performancedaten, aber auch Suchmaschinen Optimierung, Funktionen für progressive Webanwendungen, Barrierefreiheit und empfohlene Vorgehensweise ermitteln. Dabei erzeugt Lighthouse eine Punkteverteilung für jede dieser Kategorien und gibt zusätzlich Aufschluss über Optimierungsansätze. Weiter stellt Lighthouse auch eine Möglichkeit zur Simulation eines Mobilgerätzugriffs auf die Seite zur Verfügung. [LD22, S. 158 f.]

Bei der Messung mittels Lighthouse wird ein Bericht erstellt, welcher eine Wertung der Seite, nach den bereits beschriebenen Analysemöglichkeiten, beinhaltet. Teil der Performance Bewertung stellen dabei auch die Core Web Vitals dar. Die Bewertung

erfolgt in Form eines Punktesystems mit zusätzlichen Farbindikatoren. Diese fallen in drei Farben aus: grün, orange und rot. Wobei grün für eine gute Bewertung, orange für Verbesserungsbedarf und rot für eine schlechte Bewertung steht. [LD22, S. 160]

First Contentful Paint Der First Contentful Paint (FCP) beziffert wie lange es benötigt den ersten Inhalt des DOM zu laden, nachdem auf die Seite navigiert wurde. In der Tabelle 2.4 ist zu sehen wie dieser Wert in Hinsicht auf die Lighthouse Messung zu interpretieren ist. [fir]

FCP in Sekunden	**Farbcode**
0 - 1,8	Grün
1,8 - 3	Orange
Mehr als 3	Rot

Tabelle 2.4.: Time to Interactive [fir]

Time to Interactive Die Time to Interactive (TTI) misst die Dauer bist die geladene Seite vollständig Interaktiv ist. Dabei zählt die Seite als vollständig Interaktiv wenn der FCP abegeschlossen ist, alle Event Handler geladen wurden und die Reaktionsdauer von Benutzerinteraktionen der Website innerhalb von 50 Millisekunden statt findet. In der Tabelle 2.5 wird ersichtlich wie dieser Wert in Zusammenhang mit der Lighthouse Messung zu interpretieren ist. [tim]

TTI in Sekunden	**Farbcode**
0 - 3,8	Grün
3,9 - 7,3	Orange
Mehr als 7,3	Rot

Tabelle 2.5.: Time to Interactive [tim]

Dom Content Loaded Eine weitere relevante Metrik zum messen der Performance stellt der Wert DOM Content Loaded (DCL) dar. Dabei wird der Zeitpunkt bemessen, an welchem der DOM fertig geladen ist und die Ausführung von JavaScript nicht durch Stylesheets blockiert werden kann. [cri]

2.7. Fazit

Wie zu Beginn des Kapitels 2 beschrieben wurden in diesem Abschnitt die theoretischen Grundlagen dargelegt, welche für den weiteren Prozess der Arbeit relevant sind. Dabei erfolgte die Erläuterung von webbassierten Anwendung, sowie die Betrachtung von Technologien wie JavaScript und dazugehörige Frameworks. Weiter wurde auch auf Metriken für die Performancemessung von Websiten eingegangen. Diese Grundlagen dienen als Voraussetzung für die Analysen, sowie Konzepte, die im weiteren Verlauf erstellt werden.

3. Analyse und Konzeption

Dieses Kapitel setzt sich mit der Analyse aller vorhandener Strukturen auseinander um eine Problemstellung zu beschreiben und diese zu vertiefen um anschließend Optimierungsziele zu setzen und diese als Grundlage für die Konzeptionierung fortzuführen.

3.1. Analyse der bestehenden Infrastruktur

Das Budgetierungstool, auf welchem diese Arbeit aufbaut, wurde bereits als Webanwendung angelegt und umgesetzt. Dabei beruht das bereits vorhandenes System auf dem Content Management System OctoberCMS, welches unter anderem das PHP MVC Framework Laravel nutzt. An dieses System soll eine alternative Front-End Lösung angebunden werden um die Einsatzfähigkeit sowie Effizienz von JavaScript Frameworks aufzuzeigen und zu vergleichen. Der Aufgabenbereich der Anwendung erstreckt sich dabei über die Planung sowie finanzielle Verwaltung geförderter Forschungsprojekte für die Hochschule Mittweida. Dabei wurde dieses Tool angelegt um bisherige Strukturen abzulösen und eine bessere, sowie effizientere Lösung anzubieten. Da sich die Budgetplanung von Forschungsprojekten über mehrere verschiedene Anwendungen erstreckte, wobei meist eine Planung durch den jeweiligen Projektleiter und zusätzlich durch einen Sachberater durchgeführt werden musste, was in dem zusätzlichen Schritt einer Gegenkalkulation mündete. Um dies zu vermeiden sollte ein einziges Tool zum Anlegen und Verwalten dieser Projektdaten in Hinsicht auf Gehälter und Kosten erstellt werden. Anhand dessen stehen unter anderem folgende Funktionalitäten in der Umsetzung des Budgetierungstools zu Verfügung:

- **Funktion 1:** Verwendbarkeit durch mehrere Nutzer
- **Funktion 2:** Objekte Anlegen
- **Funktion 3:** Objekte Bearbeiten
- **Funktion 4:** Objekte Löschen
- **Funktion 5:** Werte an Objekte hinterlegen

Die Abbildung 3.1 zeigt die Architektur des aktuell implementierten Systems. Dabei erstreckt sich dieses über vier wesentliche Hauptbestandteile, unter anderem dem Dockermanager welcher zur Verwaltung und Ausführung aller anderen Komponenten dient. Das System untergliedert sich dabei in drei Bereiche, dem CMS welches durch einen Apache Web Server ausgeführt wird und mit der Datenhaltung, welche als MySQL Datenbank implementiert wurde, kommuniziert. Zusätzlich kann diese Datenbank mit einer weiteren Datenbankverwaltung eingesehen und bearbeitet werden. Der Webserver ist der Bestandteil der Architektur, welcher mit der Clientseite kommuniziert, also die Webanwendung an diesen überträgt.

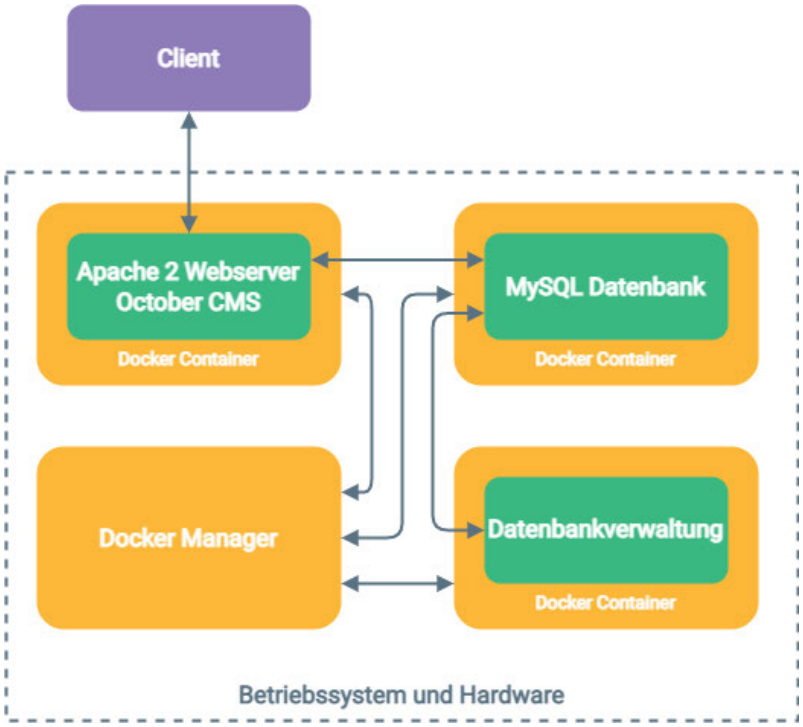

Abbildung 3.1.: Architekturskizze des vorhandenen Systems

Diese Anwendung soll als Grundlage für den Vergleich der JavaScript Frameworks dienen, dabei bleibt der strukturelle Aufbau der Architektur, sowie die serverseitige Verarbeitungslogik in Hinsicht auf Datenmanagement und Verwaltung bestehen. An der Anwendung soll die Einsatzfähigkeit der Frameworks an dem bestehend System ausgetestet, sowie durch den Austausch der Front-End Seite optimiert werden.

3.2. Analyse des bestehenden Front-Ends

Das bestehende Front-End stellt die standardmäßige Umsetzungsmöglichkeit, welche durch OctoberCMS bereitgestellt wird, dar. Technologien, die dabei die Grundlage liefern, sind unter anderem das PHP Framework Laravel und die Templating Engine Twig. Die Anwendung setzt sich daher aus mehreren Bestandteilen zusammen, welche per synchroner Datenübertragung an den Klienten übermittelt werden.

Insbesondere mit Augenmerk auf die Wahl der Datenübertragung in der momentanen Umsetzung besteht Optimierungsbedarf. Da diese auf eine synchrone Datenübertragung setzt, kann dies mit dem Wachsen der Anwendung sowie mit potentiell steigender Anzahl von Datensätzen und Nutzern dauerhaft zu Performance Problemen und hoher Serverlast führen, da bei jeder Veränderung diese Anwendung erneut vom Webserver an den Klienten übertragen werden muss.

3.3. Optimierung

Anhand der analysierten Probleme in Abschnitt 3.2 ergeben sich mehrere mögliche Lösungsansätze. Unter anderem liefert eine Alternative Umsetzung mittels Ajax eine Ausgangssituation, in der die Datenübertragung asynchron verläuft und somit, die Serverlast sinkt, insbesondere mit Augenmerk auf eine wachsende Nutzergruppe, kann diese Reduzierung Performanceverbesserungen bewirken, da die Webanwendung nur ein einziges Mal an den Klienten beim initialen Aufruf übertragen werden muss. Daher muss der Server nur spezifische Anfragen und Anforderungen von Daten verarbeiten sowie mit der Datenbank abgleichen.

3.4. Anforderungsanalyse des Front-Ends

Die wesentliche Aufgaben des Front-Ends besteht darin Daten vom Server anzufordern und anschließend zu Visualisieren, daher ergeben sich mehrere Anforderungen an den Aufbau der Anwendung, zum einen müssen alle Daten an der korrekten Position dargestellt sein, sowie eine gewisse Robustheit der Anwendung bestehen. Dazu gehört auch das Abfangen von Fehlern, so muss unter anderem der Fall abgedeckt sein, dass der Server mit der Schnittstelle für die Datenhaltung oder auch die Datenhaltung selbst, nicht erreichbar sind. Daher sollte für diesen Fall eine Rückmeldung in Form einer Alternativanzeige in der Benutzeroberfläche stattfinden.

Anhand der in Abschnitt 2.6 Vorgestellten Messkriterien bezüglich der Performance einer Webanwendung, lassen sich daraus mehrere Anforderungen für die Umsetzungen der Anwendung mittels der vorgestellten JavaScript Frameworks definieren. Um als nähere Auswahl für die Umsetzung in Frage zu kommen, sollten alle Anforderungen erfüllt werden.

Anforderung 1 Die erste Anforderung ergibt sich aus den Messungen der TTI. Diese sollte dabei einen Wert von unter 3,8 Sekunden in jeder Version der Implementierung erreichen um nach den Kriterien nach Lighthouse, als performant zu gelten.

Anforderung 2 Die zweite Anforderung soll anhand einer Messung des FCP definiert werden. Hierbei soll ebenfalls nach den vorgegebenen Kriterien ein Wert erreicht werden, welcher als performant gilt. Dafür sollte die Ladedauer nicht länger als 1,8 Sekunden betragen.

Anforderung 3 Die dritte Anforderung betrifft den Lighthousereport selbst, dabei soll jede Umsetzung eine Wertung von mindesten 90 erzielen, um als effizient genug zu gelten.

3.5. Konzeptionierung

Zur Erfüllung der Zielstellung dieser Arbeit, besteht die Notwendigkeit der Überprüfung der analysierten Optimierungsmöglichkeiten auf Verwendbarkeit mit anschließender Evaluation. Für die Implementation der untersuchten Anforderungen muss zunächst eine bestimmte Auswahl an Konzepten erstellt werden, welche die geplanten Optimierungsmöglichkeiten anhand von Technologien erläutern und aufzeigen wie die geplanten Ziele, erreicht werden sollen. Weiter soll ein Implementationskonzept kreiert und beleuchtet werden.

3.5.1. Implementationskonzept

Da das Architekturmodell der Client-Server Struktur einen Austausch des Front-Ends ermöglicht, soll daher eine komplett neue Umsetzung durch die Unterstüzung von JavaScript-Frameworks erfolgen. Dabei wird die gleiche Anwendung jeweils mit den Frameworks Vue, React und Svelte umgesetzt. Um die gleiche Ausgangssituation in allen drei Umsetzungen zu schaffen wird auf eine demonstrative Umsetzung der Anwendung gesetzt, welche einen bestimmten Funktionsumfang erhält. Die Anforderungen, die dabei entstehen sollen sich dabei auf das Abrufen von Daten aus dem Backend und deren Darstellung in der Benutzeroberfläche beschränken. Dabei sollen dem Nutzer die Projektdaten in einer simplen Oberfläche angezeigt werden.

Für die Anforderung der Daten, soll auf eine Schnittstelle, welche vom Back-End zur Verfügung gestellt wird, zurückgegriffen werden. Diese Schnittstelle bietet die Grundlage für alle Umsetzungen der Anwendung und soll zu gleichen Anteilen bei allen Implementationen zum Einsatz kommen. Diese Schnittstelle ermöglicht die Angeforderten Funktionen über HTTP-Anfragen und Antworten.

Die Implementation der JavaScript Frameworks soll dabei in verschiedenen Schritten erfolgen, als erstes soll eine Entwicklungsumgebung sowie das bestehende Back-End auf einem System geschaffen werden. Dabei soll eine Virtuelle Maschine mit einem Linux System als Server fungieren, auf welchem die Installation des October CMS sowie der SQL-Datenbank erfolgen soll. Anschließend soll das CMS mit der Datenbank verbunden werden um den Datenaustausch zu ermöglichen. Als Webserver soll die Bereitstellung eines Webserverdienstes mit Hilfe von Apache2 implementiert

werden. Damit soll zunächst die grundlegende Infrastruktur des Backends repräsentiert werden. Für die Einbindung der Frameworks soll in den nächsten Schritten eine Ordnerstruktur angelegt werden, welche die drei Umsetzungen verwaltet. Gleichzeitig soll dies auch als Entwicklungsbereich dienen welcher vom Host-System über eine SSH Tunnelung erreichbar sein soll. Als Entwicklungsumgebung soll der kostenlose Code-Editor Visual Studio Code verwendet werden, welcher ebenfalls über eine SSH-Tunnelung auf die Projekte zugreift. Um die Projekte anschließend in den Ausgangszustand für den Vergleichsprozess zu bringen, muss ein build-Prozess durchlaufen werden, anschließend werden diese in die Apache2 Webserverstruktur eingebunden, um über diesen erreichbar zu sein.

Damit der Umfang der Webanwendung in allen Umsetzungen gleich ist, soll weiter eine feste Anzahl von Komponenten, in welche sich die Anwendung aufgliedert, definiert werden. Somit soll gewährleistet werden, dass die drei Implementationen so ähnlich wie möglich aufgebaut sind, um im anschließenden Vergleich möglichst aussagekräftige Ergebnisse zu Erzielen, welche auf den gleichen Voraussetzungen beruhen. Als erster Schritt soll dabei eine Hauptkomponente, welche den Namen App trägt erstellt werden, diese soll alle weiteren Komponenten verwalten und alle Bestandteile sollen in diese Hauptkomponente eingebunden werden. Als nächstes soll eine Projektkomponente implementiert werden, diese soll alle Daten eines Projektes übergeben bekommen um diese anschließend in der Benutzeroberfläche darstellen zu können. Des weiteren soll diese Komponente für jedes Projekt, welche von der Datenbank direkt beim Laden der Seite übergeben werden erzeugt werden, somit alle Projekte in der Darstellung gelistet werden. Die Daten sollen über JavaScript Funktionen, welche auf der FetchAPI beruhen vom Server angefordert werden, hier werden Unterschiedliche Herangehensweisen gefordert, da der Featureumfang der Frameworks sich unterscheidet. So ist geplant in den Frameworks Vue und Svelte jeweils über einen Store die Daten zu beziehen und den Komponenten zur Verfügung zu stellen, bei React soll dies mithilfe der Context-Hook umgesetzt werden. Über diese Methodik soll sichergestellt sein, dass die Daten nur einmal angefordert werden müssen, sollten diese Daten in anderen Komponenten benötigt werden, kann über den Store oder den Context, auf diese zugegriffen werden. Des weiteren soll die Einbindung eines Routers erfolgen um die Seite in weitere Bestandteile aufzugliedern und zu strukturieren. Dabei soll die Erreichbarkeit der Seitenbestandteile über eine Navigation zu der jeweiligen URL ermöglicht werden.

Für die Benutzeroberfläche soll eine simple Repräsentation, welche zunächst alle relevanten Daten darstellt konzeptioniert und anschließend Implementiert werden. Für den Implementationschritt soll des weiteren auf das CCS-Framework Bootstrap zurückgegriffen werden. Diese rundimentäre Design soll den Projekttitel, sowie Notizen zu Kosten und die Kosten selbst darstellen. Weiter soll eine Alternativ Darstellung erfolgen, falls keine Daten in der Datenbank vorhanden und somit keine Projektdaten angefordert können, sowie die Mögliche Unerreichbarkeit des Backends über die HTTP-Anfragen, um derartige Ausnahmefälle abzudecken und für den Nutzer ersichtlich zu gestalten. Für die Steuerung über die verschiedenen Seitenelemente soll eine Navigationsleiste erstellt werden.

3.5.2. Vergleichskonzept

Anhand dieser Umsetzung sollen verschiedene Messungen erfolgen, welche die Ladezeiten der Anwendung ermitteln. Zum einen sollen die Messungen mit den vom Google Chrome Browser zur Verfügung gestellten Performance Insights vollzogen werden, zum anderen soll zusätzlich ein Google Lighthouse Report erstellt werden. Anschließend werden die Ladezeiten, sowie das Performance Ergebnis des Lighthouse Reports aller drei Umsetzungen miteinander verglichen, um die schnellste und performanteste Version ermitteln zu können. Weiter soll auch eine Gegenüberstellung der Projektgröße und deren Speicherplatzbedarf erfolgen um ein Fazit bezüglich des benötigten Speicherplatz bei gleichem Funktionsumfang zwischen den verschiedenen Frameworks zu ziehen. Anhand dieser Daten soll ermittelt werden, welches der Frameworks sich für die Umsetzung des Budgetierungstools am besten eignet.

Für die Ermittlung der Ladegeschwindigkeit sollen die Entwicklungstools des Chrome Webbrowsers zum Einsatz kommen. Mit diesen lassen sich verschiedene Bandbreiten simulieren und letztendlich die Übertragungsdauer der kompletten Website auslesen. Für das auslesen verschiedener Bandbreiten wird von diesem Browser eine Auswahl an verschiedenen Presets, welche verschiedene Übertragungsgeschwindikeiten darstellen, zu Verfügung gestellt. Diese sollen als vorgefertigte Maßstäbe für die Messungen eingesetzt werden, so kommen die Voreinstellungen von ”No throttling”, ”Slow 3G” und ”Fast 3G” zum Einsatz. Um eine Varianz der Werte abzudecken sollen drei Messungen je Framework und Bandbreitenpreset vollzogen und anschließend ein Mittelwert gebildet werden.

Weiter soll der Umfang von bereitgestellten Features der Frameworks in einer Gegenüberstellung verglichen werden. Um eine Abwägung für die Wahl des Frameworks anhand des standardmäßigen Funktionsumfangs ohne Zusatzinstallationen zu ermitteln. Dabei sollen die Features wie die Funktionsbasis auf Komponenten, die Bereitstellung von Databinding, die Verfügung eines Stores bzw. Context oder eines Vergleichbaren Features ermittelt werden, sowie die Unterstützung von Routing für die Navigation in Single Page Applications.

3.6. Fazit

Anhand der Analyse des bestehenden Systems, sowie des beschriebenen Implementationskonzeptes in diesem Kapitel, kann die Anwendung nun in ihren Ausführungen umgesetzt werden. Weiter ergeben sich die Grundvoraussetzungen für die Evaluation, indem Anforderungen formuliert und Vergleichskriterien definiert wurden.

4. Implementierung

Dieses Kapitel beschäftigt sich mit der Umsetzung der für den Vergleich vorgesehenen Webanwendung in ihren verschiedenen Zusammensetzungen, dabei wird auf die Vorgehensweise, den Arbeitsprozess, sowie Besonderheiten die sich bei der Entwicklung zeigten, eingegangen.

4.1. Implementation des Webservers

Die Implementierung des Webservers erfolgt auf einer Virtuellen Maschine, welche über das von Windows bereitgestellte Virtualisierungstool "Hyper-V" erstellt wird. Anschließend wird das Linux System Ubuntu in der Version 20.04.4 LTS auf der virtuellen Maschine installiert. Dieses System dient zur Bereitstellung des Webservers, sowie zur Verwaltung des CMS OctoberCMS, als auch als Datenbankserver. Auch die Erstellung und Verwaltung der Front-End Komponente, welche mittels der JavaScript Frameworks erstellt werden, findet ihre Ausführung und Verwaltung auf dieser Virtuellen Maschine.

Zunächst erfolgt die Installierung von Apache2, welches zur Bereitstellung des Webserverdienstes dient. Anschließend erfolgt die Installation von MySQL, was eine Grundvoraussetzung für alle weiteren Schritte für die Installation des CMS darstellt. Daraufhin kann der Apache Server über den Befehl aus Beispiel 4.1 gestartet werden.

```
1 $ sudo service apache2 start
```

Listing 4.1: Starten des Webservers

Anschließend werden ein Nutzer und eine Datenbank mittels SQL initialisiert. Nachdem diese Vorraussetzung geschaffen ist, folgt die Installation von OctoberCMS. Dabei musste eine Verbindung zwischen dem CMS und der Datenbank erstellt werden, da die Datenbank-Kommunikation vom CMS übernommen wird. Im nächsten Schritt

wird der Webserverdienst, sowie die Datenbank gestartet, um die Funktionalität des Back-Ends, sowie die Erreichbarkeit des Webservers und des CMS zu prüfen.

4.2. Arbeitsumgebung

Für die Entwicklung der Webanwendungen erfolgt die Erstellung einer Ordnerstruktur für die jeweilige Umsetzung mit einem der in Kapitel 2.4 Vorgestellten JavaScript Frameworks. Anschließend findet ein Kommunikationsaufbau zwischen dem Host-System der virtuellen Maschine und der virtuellen Maschine selbst mittels SSH-Tunnelung statt. Die Programmierung selbst wurd mittels des kostenlosen Code-Editors Visual Studio Code durchgeführt, welcher es erlaubt, mittels eines Plugins, über die SSH-Verbindung remote auf die Projekte zuzugreifen.

4.3. Implementation mittels React

Als erster Schritt für die Implementation der React-Umsetzung der Anwendung erfolgte die Erstellung mit Hilfe des im Grundlagenkapitel 2.3.4 vorgestellten Package Managers NPM über den Befehl *npx create-react-app*. Anschließend wurde die Funktionalität des neuen leeren Projektes mittels des Befehls *npm start* im Projektordner überprüft. Dieser Befehl startet einen Webserverdienst, welcher es ermöglicht die Seite im Entwicklungsstand im Browser aufzurufen.

Verzeichnis	Funktion
components	React Komponenten
context	React ContextAPI
pages	React Komponenten für das Routing
App.css	Stylesheet
App.js	Hauptkomponente
index.js	Hauptdatei für Rendering und Einbindung der App Komponente

Tabelle 4.1.: Übersicht und Verzeichnisse

Wie in der Übersicht 4.1 zu sehen gliedert sich das Projekt in teilweise Vorgegebene Strukturen des Frameworks, sowie eine Strukturierung, welche für die Entwicklung angelegt wird, um die Entwicklung übersichtlich und einheitlich zu gestalten. Der Ordner "components" enthält alle Komponenten, welche für die Umsetzung benötigt werden. Der "context" Ordner ist exklusiv für die Umsetzung mit React, in diesem Ordner befindet sich ein ContextProvider, das Konzept der ContextAPI wurde bereits im Grundlagenkapitel Abschnitt 2.4.1 erläutert. Dieser ContextProvider ermöglicht es mehreren Komponenten auf den gleichen Datensatz zuzugreifen.

Bevor jedoch dieser Datensatz verteilt werden kann, muss er zunächst angefordert werden. Wie im Codebeispiel 4.2 zu sehen, geschieht dies mithilfe der JavaScript-Methode "fetch", welche im Abschnitt 2.3.2 mit der Funktionsweise von AJAX vorgestellt wurde. Dabei wird ein Objekt zurückgegeben, welches die Daten im JSON-Format enthält. Dieses Objekt, welches im Beispiel durch die Konstante "data" repräsentiert ist, wird anschließend über die Methode "setData" an eine "state" Konstante übergeben.

```
1 const getProjects = async () => {
2     const response = await fetch(`http://${ip}/index.php/
    projects`);
3     const data = await response.json();
4     setData(data);
5 }
```

Listing 4.2: Asynchrone Datenanforderung mittels JavaScript

Da nun aufgezeigt wurde, wie die Anforderung der Daten vom Backend erfolgte, kann der Einsatz der Context Hook genauer betrachtet werden. Um diese im React Projekt umzusetzen wird zunächst eine dafür vorgesehene JavaScript Datei mit dem Namen "ProjectContext.js" angelegt. Wie im Beispiel 4.3 zu sehen muss dabei der ProjectContext als Context über die von React zur Verfügung gestellte Methode "createContext" initialisiert werden. Anschließend erfolgt die Deklarierung und Initialisierung des ContextProviders, welcher von React benötigt wird. Diesem ist eine Konstante, welcher die "useState" Hook zugrundelegt, zugeordnet. Der Vorteil dieser Konstante besteht darin, dass die Benutzeroberfläche automatisch angepasst wird, wenn sich ihr Wert ändert. In diesem Fall werden die Daten also automatisch überall dort in der View angepasst, wo dieser "state" behaftete Wert zum Einsatz kommt. Die Methode des ContextProviders muss im letzten Schritt die Werte, welche von anderen Komponenten genutzt werden sollen als Rückgabewert zurückgegeben werden,

was im Beispiel im *return* Statement mit dem Wert ”data” geschieht. Weiter ist auch der Einsatz der useEffect Hook zu sehen. Diese bewirkt die automatische Ausführung des Codes den sie enthält beim laden der Website. Dabei wird die Methode zur Anforderung der Daten vom Server, welche in 4.2 vorgestellt wurde ausgeführt.

```
1  const ProjectContext = createContext();
2
3  export const ProjectProvider = ({children}) => {
4      const [data, setData] = useState([]);
5
6    useEffect(() => {
7      getProjects();
8    }, [])
9
10   return <ProjectContext.Provider value = {{
11         data
12     }}>
13         {children}
14     </ProjectContext.Provider>
15 }
```

Listing 4.3: Context Hook

Für die Strukturierung des Projektes gliedert sich dieses in verschiedene React-Komponenten, jeder dieser Komponenten wird als Funktionskomponente, welche im Grundlagenkapitel Abschnitt 2.4.1 vorgestellt wurden, erzeugt. Dabei erfolgt die Erstellung einer bestimmte Anzahl an Komponenten wie in Tabelle 4.2 zu sehen. Die Komponenten sind dabei als Bausteine für die Webseite zu verstehen, wobei jede Komponente einen Teilbereich der Anwendung darstellt. Der Inhalt dieser Komponenten setzt sich aus HTML-Markup als Strukturgrundlage, JavaScript und CSS Klassen zusammen. Die CSS Klassen sind dabei vorgefertigte Klassen des Bootstrap Frameworks, zur vereinfachten Erstellung von Nutzeroberflächen. Die Datei App.js stellt in der Umsetzung die Hauptkomponente dar und repräsentiert die gesammte Webanwendung. In ihr findet die Implementation aller anderen Komponenten statt, die Teilweise ebenfalls ineinander verschachtelt sind.

Komponente	Funktion
Navbar.jsx	Komponente für Navigationsleiste
ProjectItem.jsx	Komponente für ein Datensatz eines Projektes
ProjectList.jsx	Komponente für die Verwaltung aller ”ProjektItem”
Home.jsx	Komponente für die Hauptseite und Hauptanzeige
About.jsx	Komponente für die Informationsseite der Anwendung
App.js	Hauptkomponente

Tabelle 4.2.: Übersicht React-Komponenten

Um nun die vom ContextProvider zur Verfügung gestellten Daten in einer Komponente verwenden zu können, kam die ”useContext” Hook von React zum Einsatz. Wie in Beispiel 4.4 zu sehen wird dafür eine Konstante initialisiert, welche die Werte über die Methode useContext beziehen kann, indem dieser Methode der jeweilige Context Provider, von welchem die Daten bezogen werden sollen, als Parameter übergeben wird.

```
1 function ProjectList() {
2   const {data} = useContext(ProjectContext);
3 }
```

Listing 4.4: Context Einbindung in einer Komponente

Nachdem die Daten nun in der Komponente erreichbar sind, muss diese noch an einer Position eingebunden werden um in der Darstellung der Seite im Browser auch für den Nutzer sichtbar zu sein. Dafür wird wie im Codebeispiel 4.5 über das Objekt ”data” mittels der Methode *map* iteriert, wobei für jedes gefundene Element des Objektes eine ”ProjectItem” Komponente erzeugt wird. Die Elemente stellen dabei selbst JavaScript-Objekte dar. Die ”ProjectItem” Komponente bekommt dann als Parameter die jeweiligen Werte der Eigenschaften ”name”, ”price” und ”description” der gefundenen Elemente übergeben.

```
1 <div className="d-flex justify-content-center allign-items-
    center">
2   {data.map((element) => (
3     <ProjectItem
4       name={element.name}
5       price={element.price}
6       description={element.description}
7     />
8   ))}
9 </div>
```

Listing 4.5: Context Einbindung in einer Komponente

Für die Navigation über die Webanwendung wird ein Router implementiert, dieser ermöglicht den Zugriff auf die Hauptansicht, sowie auf eine Informationsseite, welche in der Komponentenübersicht dieses Kapitels bereits dargestellt wurde. Dafür musste das Zusatzpaket "react-router-dom" wie im Grundlagenkapitel Abschnitt2.4.1 beschrieben, installiert werden.

Im letzten Schritt unterläuft diese Umsetzung dem "build" Prozess, wofür ebenfalls der Node Package Manager verwendet wird. Dafür kommt der Befehl *npm run build* zum Einsatz. Das anschließend fertige Projekt wird dann in die Apache2 Webserverstruktur eingebunden.

4.4. Implementation mittels Svelte

Für die Implementation der Svelte-Variante, muss als erstes ebenfalls ein leeres Svelte-Projekt erzeugt werden. Dafür wird das von Svelte zur Verfügung gestellte git-Repository über den Befehl *git clone https://github.com/sveltejs/svelte.git* übernommen und anschließend alle Abhängigkeiten im Projektordner mittels des NPM Befehls *npm install* installiert. Anschließend erfolgt ebenfalls ein Test der Funktionalität des neuen leeren Projektes mittels des Befehls *npm run dev* , indem es über den Entwicklungsserver ausgeführt wird.

Verzeichnis	Funktion
components	Svelte Komponenten
store	Svelte Store
pages	Svelte Komponenten für das Routing
App.svelte	Hauptkomponente
main.js	Hauptdatei für Rendering und Eibindung der App Komponente

Tabelle 4.3.: Übersicht Svelte-Namespaces und Verzeichnisse

Wie in der Tabelle 4.3 zu sehen wird die Gliederung des Projektes ähnlich zu der React Umsetzung angelegt. Ein wesentlicher Unterschied in dieser Umsetzung zum React-Projekt, stellt die Verwendung des Svelte Stores dar. Funktionell erfüllt er jedoch ähnlich zu dem React "ContextProvider" die Aufgabe, alle angeforderten Daten zentral zu verwalten, um den Zugriff auf die Daten durch alle anderen Komponenten zu ermöglichen.

Die Anforderung der Daten erfolgt analog zu dem vorgestellten Beispiel der React Implementation 4.3, jedoch anders als bei React wurde bei Svelte der Svelte Store welcher in Abschnitt 2.4.2 vorgestellt wurde für die Verwaltung und Verteilung der Daten verwendet. Dafür muss zunächst eine Variable initialisiert werden, welcher der von Svelte zur Verfügung gestellten Methode "writable" zugewiesen werden muss. Zuletzt wird dieser Variable anschließend noch die angeforderten Werte zugewiesen.

```
1 export let projects = writable([]);
2
3 const getProjects = async () => {
4     const response = await fetch(`http://${ip}/index.php/
      vehicles`);
5     const data = await response.json();
6     projects.set(data);
7 }
```

Listing 4.6: Context Einbindung in einer Komponente

Um die Angeforderten Daten nun in die gewünschte Komponente zu integrieren, muss diese, wie in Beispiel 4.7 vom "store" importiert werden. Anschließend kann diese Variable für alle weiteren Schritte in der Komponente verwendet werden.

```
1 import {projects} from "../../store/stores.js"
```

Listing 4.7: Context Einbindung in einer Komponente

Um die Daten automatisch beim Laden der Website anzuforden, werden die Funktionen, in der von Svelte zur Verfügung gestellten Methode *onMount* aufgerufen. Diese Methode wird beim Zugriff auf die Website ausgeführt.

```
1 import { onMount } from "svelte";
2
3   onMount(() => {
4     getProjects();
5   });
```

Listing 4.8: Context Einbindung in einer Komponente

Um in dieser Realisierung ebenfalls das Routing parallel zu der React Implementation zu ermöglichen muss das Zusatzmodul ”svelte-spa-router” installiert und in die Projektstruktur eingebunden werden.

Abschließend unterläuft diese Umsetzung ebenfalls dem ”build” Prozess analog zur der React Umsetzung 4.3 Das anschließend fertige Projekt wird dann in die Apache2 Webserverstruktur eingebunden.

4.5. Implementation mittels Vue

Auch dieses Projekt wird wieder mittels NPM erzeugt, allerdings erfolgte dies wieder unter einer anderen Befehlskette. Für die Initialisierung des Vue Projektes kommt der Befehl *npm init vue@latest* zum Einsatz. Anschließend muss eine Reihe von Konfigurationen abgeschlossen werden, welche unter anderem die mögliche Installationen von Zusatzmodulen enthält. Diese Module sind jedoch für dieses Projekt irrelevant.

Verzeichnis	Funktion
components	Vue Komponenten
store	Vue Store
pages	Vue Komponenten für das Routing
App.vue	Hauptkomponente
main.js	Hauptdatei für Rendering und Einbindung der App Komponente

Tabelle 4.4.: Übersicht Vue-Namespaces und Verzeichnisse

Die Projekt Struktur ist dabei nahezu identisch zu der, der Svelte Umsetzung, mit dem Unterschied, dass die Komponenten hier als ".vue" Datei bestehen. Auch der Aufbau und die Anzahl der Komponenten orientiert sich an den vorherigen Umsetzungen. So finden sich auch hier die Komponenten "ProjectItem", "ProjectList" und "Navbar" wieder.

Auch bei dieser Umsetzung erfolgt die Anforderung der Daten vom Backend analog zu den Umsetzungen von Svelte und React über die JavaScript *fetch* Methode. Ähnlich zur Svelte Umsetzung geschieht dies im "store", auf welchen ebenfalls durch alle anderen Komponenten zugegriffen werden kann. Da Vue standardmäßig keine vergleichbare Funktion zur ContextHook von React und dem Svelte Store besitzt, wird die Erweiterung Vuex zusätzlich installiert. Die Funktionsweise des Vuex Stores wurde bereits im Grundlagenkapitel 2.4.3 erläutert. Zu Verwendung in der Implementation musste dafür das Vuex Paket über den befehl *npm install vuex@next* installiert werden. Um diesen Vuex Store in der Anwendung nutzen zu können muss zunächst die Methode *createStore* angewandt werden, wie in Codebeispiel 4.9 dargestellt. Diese erhält ein Objekt als Übergabeparameter mit verschiedenen Konfigurationen. Die "actions" können asynchrone Funktionen enthalten, in diesem Fall wird über die asynchrone JavaScript Methode die Daten vom Server anfordert. Die "mutations" dienen zur Veränderung der "state" Werte, in dem Fallbeispiel werden dabei die in den "mutations" die vorher angeforderten Werte dem "state" Array "projects" zugewiesen. Die "getters" dienen zur Anforderung der "state" Werte in einer Komponente.

```
1  import { createStore } from 'vuex'
2
3  export default createStore({
4    state: {
5      projects: []
6    },
7    getters: {
8      allProjects: state => state.projects
9    },
10   mutations: {
11     GET_PROJECTS: (state, project) => {
12       state.projects = projects
13     }
14   },
15   actions: {
16     async getProjects ({ commit }) {
17       const response = await fetch(`http://${ip}/index.php/
       vehicles`);
18       const data = await response.json();
19     commit('GET_PROJECTS', data)
20     }
21   },
22   modules: {
23   }
24 })
```

Listing 4.9: Context Einbindung in einer Komponente

Anschließend erfolgt der Import der Getter in der "ProjectList"-Komponente wo über die *v-for* Direktive über alle Daten itteriert wird und für jede gefundenen Eintrag, eine Tabellenzeile erzeugt wird.

Um in dieser Umsetzung wie in allen vorherigen Implementationen ebenfalls die Funktionalität des Routings einzubinden muss das Zusatzpaket "vue-router" installiert werden.

Auch diese Implementierung muss den "build" Prozess durchlaufen um das anschließend fertige Projekt in die Apache2 Webserverstruktur einzubinden.

4.6. Implementation des User-Interface

Die Benutzeroberfläche wird optisch in allen drei Umsetzungen identisch implementiert und setzt sich dabei aus den Komponentenbereichen, wie bereits in der vorherigen Abschnitten, erläutert zusammen. Wie in Abbildung 4.1 zu sehen ermöglicht die Oberfläche die Auswahl einer Buchung, anschließend werden alle dazugehörigen Kostenpunkte in Tabellenform aufgelistet. Weiter wird eine Navigationsleiste um über Seite unter der Verwendung der Router navigieren zu können, dargestellt. Für die optische Gestaltung, wird das CSS-Framework Bootstrap verwendet. Dies ermöglichte durch vorgefertigte CSS-Klassen eine schnelle Erstellung eben dieses User-Interfaces.

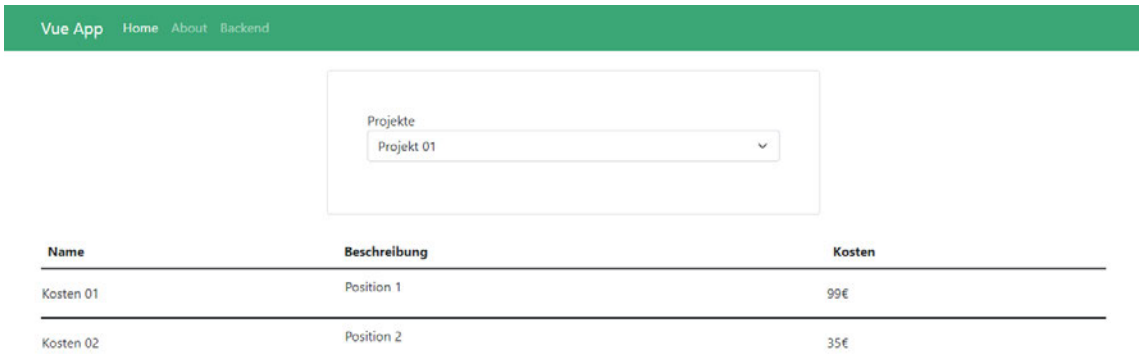

Abbildung 4.1.: Benutzeroberfläche

4.7. Fazit

Innerhalb dieses Kapitels wurde, wie zu Beginn des Abschnitts beschrieben, die Implementation der Webanwendung in ihren verschiedenen Bestandteilen sowie Umsetzungen erläutert. So wurde auf die dahinter liegende Systemgrundlage mit allen Grundvoraussetzungen eingangen, sowie die Arbeitsumgebung beschrieben. Im weiteren Verlauf wurde auf den Kernbestandteil der gesamten Implementation, den drei Umsetzungen mit den jeweiligen JavaScript-Frameworks React, Svelte und Vue eingegangen. Im weiteren dienen die drei Ausführungen als Grundlage für die Evaluation anhand der in Abschnitt 3.5.2 erläuterten Vergleichskriterien.

5. Evaluation

Auf die in Kapitel 4 ausgeführte Implementation folgt nun die Evaluation, anhand des in Abschnitt 3.5.2 dargelegten Vergleichkonzeptes. Mit dessen sollen verschiedene Messungen durchgeführt und ausgewertet, sowie weitere Kennziffern für den Vergleich dazugezogen werden. Dabei soll, wie bereits in der Zielstellung erläutert primär die effizienteste Umsetzung ermittelt werden.

5.1. Verbindung der JavaScript FrontEnds mit OctoberCMS

Zunächst soll die Anforderung der Daten in der neuen Umsetzung des Front-Ends von dem bereits bestehenden System, welches mittels OctoberCMS umgesetzt wurde, evaluiert werden. Der dafür Entscheidende Faktor stellt die Implementationsmöglichkeit aller Umsetzungen in das bestehende System dar.

Um diesen Test durchzuführen, wurde jede der Umsetzungen innerhalb eines frei gewählten Webbrowsers geöffnet. Anschließend navigiert der Nutzer auf das Drop-Down Menü, um ein Projekt auszuwählen. Dies ermöglicht das Laden der dazugehörigen Buchungsinformationen in die Benutzeroberfläche.

Nach Abschluss der beschriebenen Schritte sollte die testende Person folgendes Ergebnis erwarten. Nachdem die Auswahl vollzogen wurde, sollte bei akkurater Anfrage an den Webserver alle zugehörigen Buchungsinformationen zu einem Projekt in Tabellenform erscheinen. Die Informationen sollten dabei an richtiger Position, welche sich in Name, Beschreibung und die Kosten aufgliedern, in der Tabelle eingeordnet sein. Um zu Überprüfen ob die Daten tatsächlich korrekt sind, besteht die Möglichkeit sich im Backend des CMS einzuloggen und die dort hinterlegten Daten mit denen des Front-Ends zu vergleichen.

5.2. Lighthouse Report

Dieser Abschnitt betrachtet die Ergebnisse des Lighthouse Reports an den Umsetzungen des Budgetierungstools. Anhand der in Abschnitt 3.4 definierten Anforderungen an die Projekte, sollen die Ergebnisse dieser Reports evaluiert werden. Dafür wurde die Webanwendung im Browser aufgerufen und anschließend mit den Entwicklerwerkzeugen die Analyse durchgeführt.

5.2.1. React

Die Ergebnisse, die der Lighthouse Report bei React erzielte werden aus der Abbildung 5.1 ersichtlich. Unter Anbetracht der Anforderung 1 lässt sich feststellen, dass diese Umsetzung mit einer TTI von 0,8 Sekunden diese Anforderung deutlich erfüllt. Auch die zweite Anforderung, einen FCP von unter 1,8 Sekunden zu erzielen, wurde mit einem gemessenen FCP von 0,2 Sekunden eingehalten. Die Gesamtwertung der Performance mit einem Punktwert von 100 übererfüllt den in Anforderung 3 festgelegten Mindestpunktwert.

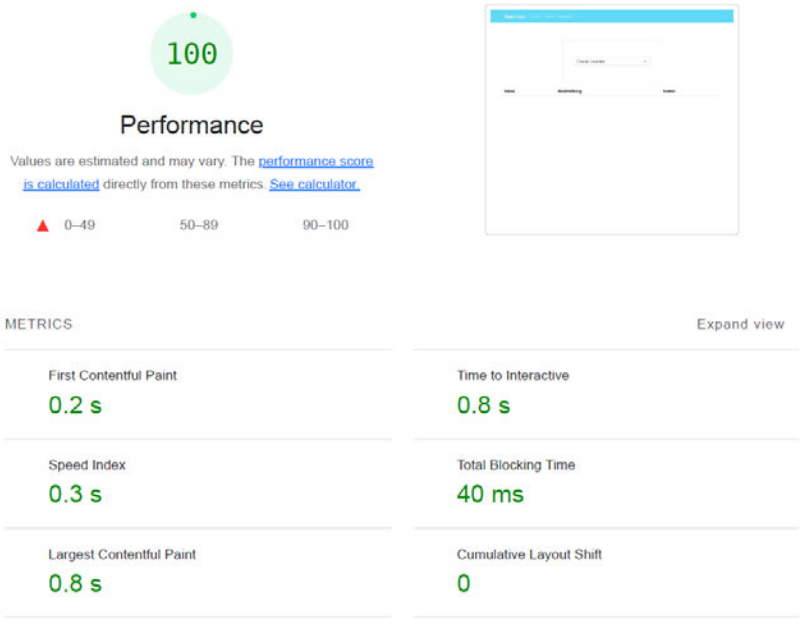

Abbildung 5.1.: Lighthouse Report - React

5.2.2. Svelte

Bei der Umsetzung mit Svelte wurden folgende Resultate, wie in Abbildung 5.2 zu erkennen erzielt. Die erste Anforderung wurde mit einer TTI von 0,4 Sekunden erfüllt. Als nächtes erfolgte die Betrachtung des FCP, welcher mit 0,4 Sekunden auch der zweiten Anforderung an die Anwendung entspricht. Für die dritte und letzte Anforderung in der Performancebewertung, erzielte Svelte eine Punktwertung von 100, wodurch diese Anforderung ebenfalls als erfüllt zu betrachten ist.

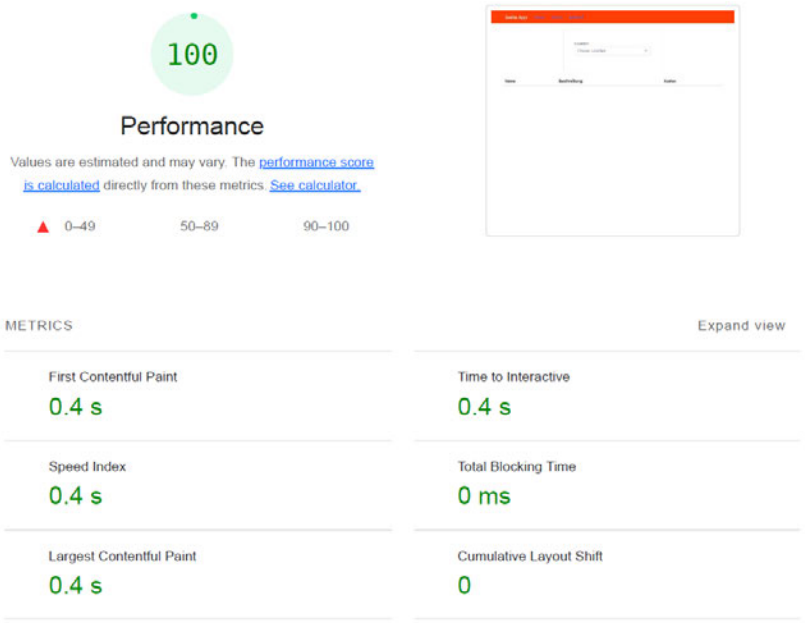

Abbildung 5.2.: Lighthouse Report - Svelte

5.2.3. Vue

Bei der dritten Umsetzung des Budgetierungstools mit dem Framework Vue konnten die definierten Anforderungen ebenfalls erfüllt werden. Dabei erzielte diese Version für die erste Anforderung einen TTI Wert von 0,4 Sekunden. Der Messwert des FCP für die zweite Anforderung ergab eine Dauer von 0,3 Sekunden. Der abschließende Punktwert der Performance erzielte 100 Punkte, und entsprach damit auch der dritten Anforderung.

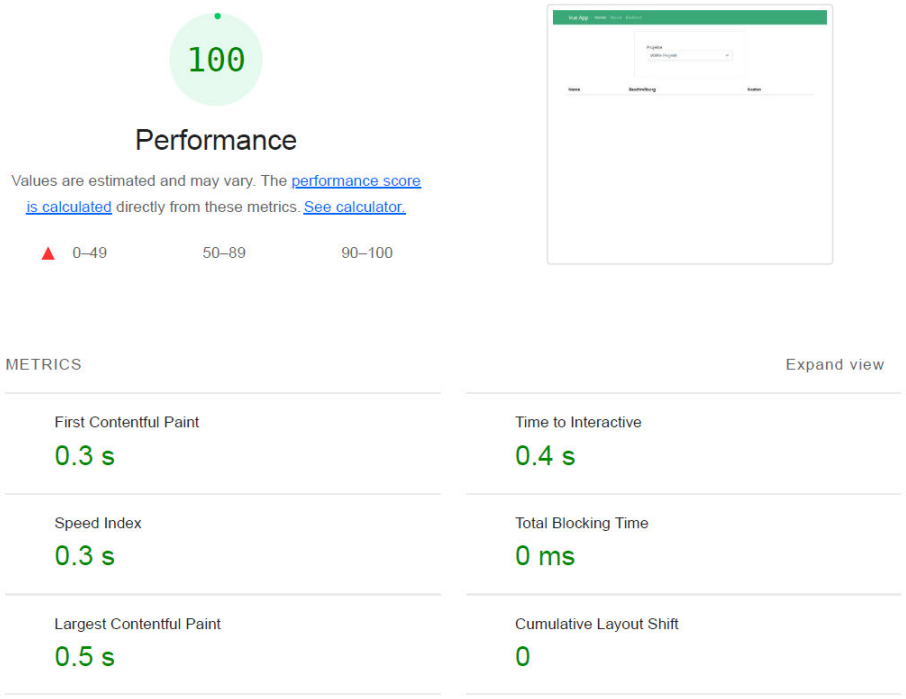

Abbildung 5.3.: Lighthouse Report - Vue

5.3. Vergleich der JavaScript-Frameworks

Dieser Abschnitt beschäftigt sich mit dem Vergleich der JavaScript Frameworks durch die in Abschnitt 3.5.2 vorgestellten Vergleichskriterien.

5.3.1. Test der Übertragungsgeschwindigkeit

Nachdem wie in der Implementation erläutert, die Projekte in die Apache2 Webserverstruktur eingebunden wurden, kann nun, mittels der Entwicklungstools des Webbrowsers die Übertragungsdauer für die jeweilige Umsetzung ermittelt werden. Dabei stehen vier Testfälle zu je drei unterschiedlichen Übertragungsgeschwindigkeiten zur Verfügung. Die Durchführung der Testfälle wird dabei beispielhaft beschrieben und die erzielten Ergebnisse verglichen und ausgewertet.

Die Durchführung dieser Testfälle setzt dabei beim Aufruf der Website im Browser an. Gemessen wurde dabei jeweils die in Abschnitt 2.6 erläuterten Kriterien DCL, FCP, TTI und LCP. Die Gegenüberstellung erfolgt dabei Anhand der kalkulierten Mittelwerte, welche sich aus den Messungen ergaben. Die Ausführlichen Messungen befinden sich im Anhang A.

Framework	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
React	0,22	0,28	0,28	0,28
Svelte	0,24	0,24	0,24	0,24
Vue	0,23	0,24	0,24	0,24

Tabelle 5.1.: Mittelwerte Ladegeschwindigkeit - No Throttling

Wie Anhand der Tabelle 5.1 zu sehen, unterscheiden sich die Ladegeschwindigkeiten der verschiedenen Umsetzungen nur geringfügig, dennoch erwies sich dabei Vue als schnellste Umsetzung, zwar brauchte diese für das Laden des DOM-Trees länger als React, allerdings wurden alle weiteren Zusatzmodule wie die Stylesheets und Skripte schneller geladen. Svelte erzielte das stabilste Ergebnis in Hinsicht auf Ladedauer zwischen des DOMs und aller Zusatzinhalte. React wies eine erhöhte Diskrepanz zwischen der Ladedauer des DOM-Trees und dem laden aller weiteren Inhalte.

Framework	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
React	0,83	0,90	0,90	0,90
Svelte	0,79	0,80	0,80	0,80
Vue	0,77	0,78	0,78	0,78

Tabelle 5.2.: Mittelwerte Ladegeschwindigkeit - Fast 3G

Auch bei der Veränderung der Bandbreitenkonfiguration zu ”Fast 3G” wies Vue, wie in Tabelle 5.2 veranschaulicht, im Mittel die schnellste Ladezeit auf, mit geringem Abstand dazu folgte Svelte. React wiederum wies auch wie bei der vorherigen Messung 5.1 einen erhöhten Abstand zwischen dem Laden des DOM-Trees und aller zusätzlichen Inhalte auf.

Framework	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
React	2,24	2,31	2,31	2,31
Svelte	2,23	2,23	2,23	2,23
Vue	3,58	3,59	3,59	3,59

Tabelle 5.3.: Mittelwerte Ladegeschwindigkeit - Slow 3G - Messung 1

Bei den Messung mittels der Konfiguration ”Slow 3G” fällt, wie in Tabelle 5.3 zu sehen, auf, dass Vue einen erhebliche längere Ladezeit im Vergleich zu den anderen Umsetzungen aufweist. Bei den Messungen der Umsetzung mit dem Framework Vue gab es eine Spitze, welche deutlich von den vorherigen Messungen abwich, aufgrund dessen erfolgte eine erneute Messung, um möglichst einen aussagekräftigeren Wert zu erhalten.

Framework	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
React	2,24	2,31	2,31	2,31
Svelte	2,23	2,23	2,23	2,23
Vue	2,19	2,19	2,19	2,19

Tabelle 5.4.: Mittelwerte Ladegeschwindigkeit - Slow 3G - Messung 2

Nach erneuter Durchführung der Messung ist zu erkennen, wie aus Tabelle 5.4 ersichtlich, dass die Abweichung deutlich geringer ausfällt. Auch in dieser Konfiguration erwies die Umsetzung mittels des Framworks Vue, das Ergebnis mit der geringsten Ladedauer in allen Bereichen. Die Version, welche auf Svelte beruht, lieferte das zweit schnellste Ergebnis. Die Tendenz, die React bereits bei den vorherigen Messungen aufwies, verdeutlicht sich ebenfalls wieder in dieser Messung, durch eine erhöhte Differnz zwischen dem DCL und allen weiteren Messkritieren.

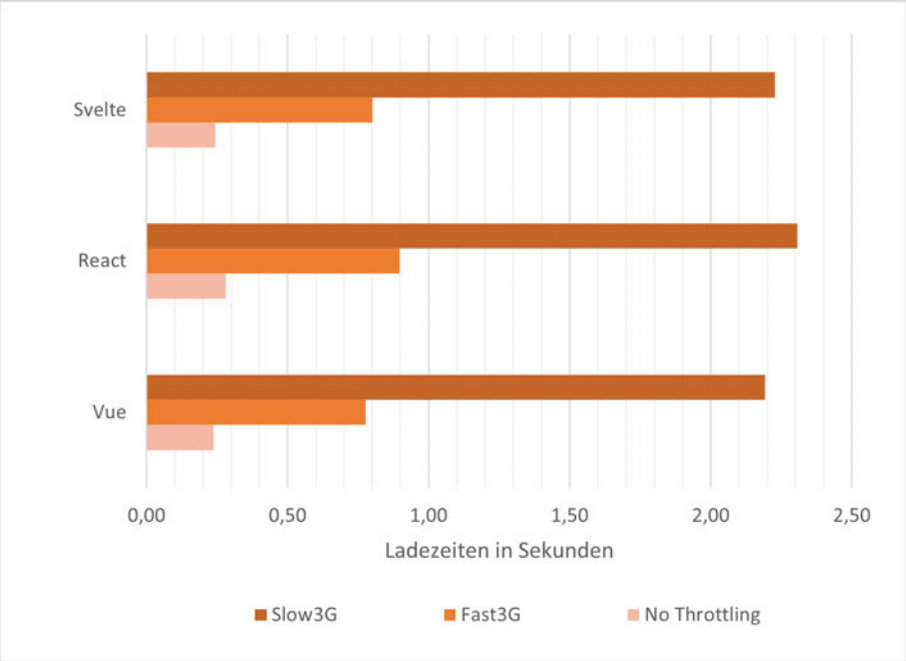

Abbildung 5.4.: Vergleich aller Ladegeschwindigkeiten der Frameworks

Abschließend wurden die Werte wie in Abbildung 5.4 zu sehen als Balkendiagramm zusammengefasst. Dabei lässt sich erkennen, dass Vue bei allen Messung, mittels der Performance Insights, die geringste Ladezeit aufweist. Mit einem geringen Unterschied folgt Svelte, wobei die Ladegeschwindigkeiten nahezu identisch zu Vue ausfallen. Ein größerer Unterschied bei allen drei Messungen ist bei der Implementation mit React zu erkennen, auch wenn dieser nur im Bereich von einer Zehntelsekunde auftritt.

5.3.2. Vergleich der Lighthouse Resultate

Anhand der in Abschnitt 5.2 durchgeführten Lighthouse Reports zu den jeweiligen Umsetzungen. lässt sich feststellen, dass alle Versionen die Höchstpunktzahl unter dem Kriterium der Performance erlangten. Daher entsteht der Entschluss, das unter Anbetracht dieses Vergleichskriterium die drei Implementationen als gleichwertig zu erachten sind.

5.3.3. Vergleich der Projektgrößen

Um den Speicherplatzbedarf der jeweiligen Umsetzungen des Budgetierungstools zu messen, wurde der File Explorer des Host Systems genutzt, dabei ergaben sich wie in Tabelle 5.5 zu sehen folgende Werte:

Framework	**Projektgröße in MB**	**Bundlegröße in MB**
React	250,00	2,10
Svelte	35,30	0,79
Vue	142,80	1,80

Tabelle 5.5.: Messwerte der Projektgrößen

Beim Vergleich der Projektgrößen fällt auf, dass die Umsetzung mit dem Framework Svelte mit Abstand den geringsten Speicherplatzbedarf, bei gleichem Funktionsumfang aufweist. Gefolgt von der Umsetzung mittels Vue. React erwies sich als die Umsetzung mit dem größten Speicherbedarf.

5.3.4. Vergleich des standardmäßigen Funktionsumfangs

Anhand der erläuterten Funktionen der JavaScript Frameworks im Grundlagenkapitel 2.4 kann in diesem Abschnitt der Funktionsumfang der Frameworks darauf verglichen werden, welche Möglichkeiten ohne die Installation von Zusatzmodulen bereitsteht.

Funktion	**React**	**Svelte**	**Vue**
Komponentenbasiert	Ja	Ja	Ja
Data Binding	Ja	Ja	Ja
Context oder Store	Ja	Ja	Nein
Routing	Nein	Nein	Nein

Tabelle 5.6.: Standardmäßiger Funktionsumfang der Frameworks

In der Tabelle 5.6 wird ersichtlich, dass der standardmäßige Funktionsumfang von Svelte und React anhand der betrachteten Kriterien gleich ist. Da es sich bei allen drei Frameworks um Komponenten basierte JavaScript Frameworks handelt, wird dieser Funktionsumfang auch von allen im Standardumfang unterstützt. Für die Implementation eines Stores musste für die Umsetzung mit dem Framework Vue, das Zusatzmodul Vuex installiert werden, anders als bei den Frameworks Svelte und React wo eine ähnliche Funktionsweise standardmäßig bereitgestellt wird. Für die Verwendung eines Routers in den umgesetzten Single Page Applications, musste für alle drei Frameworks auf die Installation eines Zusatzmoduls zurückgegriffen werden.

5.4. Fazit

Wie zu Beginn des Kapitels definiert, wurde innerhalb dieses Abschnittes die Evaluation der drei Umsetzungen des Systems beschrieben. Dabei konnten alle definierten Kriterien erfolgreich geprüft und gemessen werden. Die Anforderungen die dabei an das System gestellt wurden, konnte von allen Implementationen erfüllt werden. Auf Basis der dargelegten Kriterien für die Umsetzung der Front-End Komponente des Budgetierungstools lässt sich anhand des geringen Speicherplatzbedarf, sowie des umfangreichen Funktionsumfangs und der geringen Übertragungsdauer, die Empfehlung für das JavaScript Framework Svelte, für eine derartige Umsetzung, geben. Da sich die Umsetzungen mit React und Vue ebenfalls als effizient erwiesen haben, wenn auch nicht wie Svelte, ist eine Verwendung dieser Frameworks hinsichtlich der Performance sowie durch die Möglichkeit über Zusatzmodule den gleichen Funktionsumfang zu erlangen, grundsätzlich empfehlenswert.

6. Zusammenfassung und Ausblick

Dieses Kapitel dient als finale Betrachtung dieser wissenschaftlichen Arbeit, wobei die erworbenen Ergebnisse zusammengefasst rekapituliert, sowie eine abschließende Aussage zum Erfolg dieser Arbeit getroffen werden soll. Weiter sollen mögliche Weiterführungen, wofür diese Arbeit als Grundlage dienen kann, in Ausblick gestellt werden.

6.1. Zusammenfassung

Am Anfang dieser Arbeit wurde in Kapitel 1 als Ziel dieser Arbeit definiert, dass die Einbindung einer neuen Front-End Lösung in das bereits bestehende Budgetierungstool vollzogen werden soll. Diese Lösung sollte mittels drei verschiedener JavaScript-Frameworks implementiert werden, um anschließend eine Gegenüberstellung dieser drei Umsetzungen, anhand vordefinierter Kriterien durchzuführen. Dafür musste zunächst eine Wissensgrundlage über die verwendeten Technologien, den Frameworks und deren Funktionsumfang vermittelt werden. Folgend darauf wurde eine Analyse vollzogen, sowie ein Implementationskonzept und Vergleichskonzept erstellt.

Anschließend konnte die Webanwendung anhand des Implementationskonzeptes in ihren drei Ausführungen umgesetzt und in das bestehende System Eingebunden werden. Dafür wurde eine virtuelle Maschine erzeugt in welcher zum einen die Installation des Webserverdienstes Apache erfolgte, aber auch in Installation von OctoberCMS, einer MySQL Datenbank und die Entwicklung dieser Umsetzung stattfand.

Darauf liesen sich die Implementationen Anhand der erstellten Anforderungsanalyse und des Vergleichkonzeptes evaluieren. Beginnend mit den Performance Anforderungen, welche anhand des Lighthouse Reports bemessen wurden, konnte festgestellt werden, das alle Umsetzungen den Anforderungen gerecht werden konnten. Anschließend erfolgte eine direkte Gegenüberstellung der Frameworks, wobei sich die Umsetzung mit Svelte am effizientesten erwies.

6.2. Fazit und Ausblick

Das Ziel dieser Arbeit war es, wie in Kapitel 1 dargelegt, die effizienteste Front-End Umsetzung des webbasierten Budgetierungstools, unter Verwendung der drei JavaScript Frameworks React, Svelte und Vue, zu ermitteln. Abschließend ist Festzuhalten, dass die Performanceanforderungen von allen Implementationen erfüllt werden konnte, im Direktvergleich erwies sich die Umsetzung mit Svelte, anhand der festgelegten Kriterien, als effizienteste Lösung. Somit kann die Forschungsfrage als erfüllt betrachtet werden.

Dennoch besteht die Möglichkeit in Folgeprojekten an die Thematik weiter heranzutreten. Die exemplarischen Umsetzungen spiegeln lediglich den Vergleich eines Anwendungsbereiches dar. Somit könnten künftig sowohl weitere Messkriterien als auch weitere Anwendungsbeispiele, sowie ein erweiterter Funktionsumfang, für einen derartigen Vergleich herangezogen werden.

Literaturverzeichnis

[AB20] Roy Derks Adam Boduch: *React and React Native A complete hands-on guide to modern web and mobile development with React.js*, Packt Publishing Ltd., 3 Aufl., 4 2020, ISBN 978-1-83921-114-0.

[Abt19] Dietmar Abts: *Masterkurs Client/Server Programmierung mit Java*, Springer Vieweg, Wiesbaden, 5 Aufl., 3 2019, ISBN 978-3-658-25924-2.

[Ack21] Philip Ackermann: *JavaScript : Das umfassende Handbuch*, Rheinwerk Verlag, 3 Aufl., 9 2021, ISBN 9783836286299.

[Att20] Joe Attardi: *Modern CSS*, Berkeley, CA Apress, 1 Aufl., 2020, ISBN 9781484262948.

[Ben14] Günther Bengel: *Grundkurs Verteilte Systeme Grundlagen und Praxis des Client-Server und Distributed Computing*, Wiesbaden Springer Vieweg, 4 Aufl., 2014, ISBN 9783834821508.

[cri] *Measuring the Critical Rendering Path*, URL: https://web.dev/critical-rendering-path-measure-crp/, besucht am 01.11.2022.

[Dei22] Fabian Deitelhoff: *Vue.js Von Grundlagen bis Best Practices*, dpunkt.verlag, 08 2022, ISBN 9783969107607.

[Dig21] IONOS Digitalguide: *Was ist eine Web-App? Definition und Web-App-Beispiele*, 3 2021, URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/verschiedene-app-formate-was-ist-eine-web-app/, besucht am 12.05.2022.

[fir] *First Contentful Paint*, URL: https://web.dev/first-contentful-paint/, besucht am 01.11.2022.

[Fre19] Adam Freeman: *Pro React 16*, New York Apress, 2019, ISBN 9781484244517.

[Gol14] Joachim Goll: *Architektur- und Entwurfsmuster der Softwaretechnik Mit lauffähigen Beispielen in Java*, Wiesbaden Springer Vieweg, 2 Aufl., 2014, ISBN 9783658055325.

[GW17] Mircea Diaconescu Gerd Wagner: *Web Applications with Javascript or Java*, De Gruyter Oldenbourg, 12 2017, ISBN 9783110499933.

[JDF+14] Baumann Joachim, Ardnt Daniel, Engelen Frank, Hardy Frank und Carsten Mjartan: *Vaadin Der kompakte Einstieg für Java-Entwickler*, dpunkt.verlag, 11 2014, ISBN 9783864915857.

[Kra20] Jörg Krause: *Introducing Bootstrap 4*, Berkeley, CA Apress, 2 Aufl., 2020, ISBN 9781484262030.

[LBR+19] Coulson Lewis, Jephson Brett, Larsen Rob, Park Matt und Zburlea Marian: *The HTML and CSS Workshop A New, Interactive Approach to Learning HTML and CSS*, 1 Aufl., 11 2019, ISBN 9781838828134.

[LD22] Samuel Larsen-Disney: *Elevating React Web Development with Gatsby*, Packt Publishing, 1 2022, ISBN 9781800202962.

[Lib22] Alex Libby: *Practical Svelte Create Performant Applications with the Svelte Component Framwork*, Berkeley, CA Apress, 1 Aufl., 2022, ISBN 978-1-4842-7374-6.

[MDN20] MDN: *JavaScript*, 12 2020, URL: https://developer.mozilla.org/de/docs/Web/JavaScript, besucht am 12.05.2022.

[MDN21] MDN: *HTML: HyperText Markup Language | MDN*, 4 2021, URL: https://developer.mozilla.org/de/docs/Web/HTML, besucht am 12.05.2022.

[NPM] NPM: *About npm | npm Docs*, URL: https://docs.npmjs.com/about-npm, besucht am 18.05.2022.

[octa] *Features - October CMS*, URL: https://octobercms.com/features, besucht am 16.08.2022.

[octb] *october/README.md at develop · octobercms/october*, URL: https://github.com/octobercms/october/blob/develop/README.md, besucht am 16.08.2022.

[Ran21] R.S. Rana: *What is the difference between Library vs Framework?*, 5 2021, URL: https://dev.to/rohitrana/what-is-the-difference-between-library-vs-framework-174n, besucht am 02.08.2022.

[reaa] *Bedingte Darstellung*, URL: https://de.reactjs.org/docs/conditional-rendering.html, besucht am 18.08.2022.

[reab] *Handhabung von Events*, URL: https://de.reactjs.org/docs/handling-events.html, besucht am 18.08.2022.

[reac] *Komponenten und Props*, URL: https://de.reactjs.org/docs/components-and-props.html, besucht am 18.08.2022.

[Rib20] Heitor Ramon Ribeiro: *Vue.js 3 Cookbook*, Packt Publishing Ltd., 2020, ISBN 978-1-83882-622-2.

[Rod20] G. Roden: *Was man über React wissen sollte*, 11 2020, URL: https://www.heise.de/blog/Was-man-ueber-React-wissen-sollte-4966420.html, besucht am 17.08.2022.

[Roh18] Matthias Rohr: *Sicherheit von Webanwendungen in der Praxis*, Wiesbaden Springer Vieweg, 2 Aufl., 3 2018, ISBN 978-3-658-20145-6.

[Sch12] Alexander Schill: *Verteilte Systeme Grundlagen und Basistechnologien*, Wiesbaden Springer Vieweg, 2 Aufl., 2012, ISBN 9783642257964.

[Seg20] Alessandro Segala: *Svelte 3 Up and Running*, Packt Publishing Ltd., 8 2020, ISBN 978-1-83921-362-5.

[Smi15] Ben Smith: *Beginning JSON*, Apress Berkeley, CA, 1 Aufl., 2015, ISBN 978-1-4842-0203-6.

[Ste14] Ralph Steyer: *JAVASCRIPT Die universelle Sprache zur Web-Programmierung*, Carl Hanser Verlag GmbH & Co. KG, 2014, ISBN 978-3-446-43947-4.

[Ste16] Ralph Steyer: *WordPress Einführung in das Content Management System*, Wiesbaden Springer Vieweg, 2016, ISBN 978-3-658-12830-2.

[Ste18] Ralph Steyer: *jQuery das universelle JavaScript-Framework für das interaktive Web und mobile Apps*, München Hanser, 2 Aufl., 2018, ISBN 9783446456518.

[Ste19] Ralph Steyer: *Webanwendungen erstellen mit Vue.js*, Wiesbaden Springer Vieweg, 2019, ISBN 9783658271701.

[tim] *Time to Interactive*, URL: https://web.dev/interactive/?utm_source=lighthouse&utm_medium=devtools, besucht am 01.11.2022.

[Vor08] Deepak Vorah: *Ajax in Oracle JDeveloper*, Springer Berlin Heidelberg, 2008, ISBN 9783540775966.

[vuea] *Components Basics | Vue.js*, URL: https://vuejs.org/guide/essentials/component-basics.html, besucht am 02.11.2022.

[vueb] *What is Vuex? | Vuex*, URL: https://vuex.vuejs.org/#what-is-a-state-management-pattern, besucht am 11.08.2022.

[Wil99] Erik Wilde: *World Wide Web Technische Grundlagen*, Berlin, Heidelberg : Springer Berlin Heidelberg : Imprint: Springer, 1 Aufl., 1999, ISBN 3-642-59944-3.

[Wol21] Jürgen Wolf: *HTML und CSS*, Rheinwerk Verlag, 4 Aufl., 7 2021, ISBN 9783836281195.

Anhang

A. Evaluationsdokumente

A.1. Messdaten der Übertragungsgeschwindigkeit No Throttling

Vue	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,24	0,24	0,24	0,24
	0,21	0,22	0,22	0,22
	0,24	0,25	0,25	0,25
Mittelwert:	0,23	0,24	0,24	0,24

Tabelle A.1.: Messwerte Ladegeschwindigkeit Vue - No Throttling

React	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,23	0,29	0,29	0,29
	0,21	0,28	0,28	0,28
	0,22	0,28	0,28	0,28
Mittelwert:	0,22	0,28	0,28	0,28

Tabelle A.2.: Messwerte Ladegeschwindigkeit React - No Throttling

Svelte	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,23	0,24	0,24	0,24
	0,25	0,25	0,25	0,25
	0,23	0,24	0,24	0,24
Mittelwert:	0,24	0,24	0,24	0,24

Tabelle A.3.: Messwerte Ladegeschwindigkeit Svelte - No Throttling

A.2. Messdaten der Übertragungsgeschwindigkeit Fast 3G

Vue	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,75	0,75	0,75	0,75
	0,81	0,81	0,81	0,81
	0,76	0,77	0,77	0,77
Mittelwert:	0,77	0,78	0,78	0,78

Tabelle A.4.: Messwerte Ladegeschwindigkeit Vue - Fast 3G

React	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,80	0,86	0,87	0,87
	0,89	0,95	0,95	0,95
	0,80	0,88	0,88	0,88
Mittelwert:	0,83	0,90	0,90	0,90

Tabelle A.5.: Messwerte Ladegeschwindigkeit React - Fast 3G

Svelte	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	0,78	0,79	0,79	0,79
	0,81	0,81	0,81	0,81
	0,79	0,80	0,80	0,80
Mittelwert:	0,79	0,80	0,80	0,80

Tabelle A.6.: Messwerte Ladegeschwindigkeit Svelte - Fast 3G

A.3. Messdaten der Übertragungsgeschwindigkeit Slow 3G

Vue	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	2,23	2,24	2,24	2,24
	2,24	2,24	2,24	2,24
	6,26	6,28	6,28	6,28
Mittelwert:	3,58	3,59	3,59	3,59

Tabelle A.7.: Messwerte Ladegeschwindigkeit Vue - Slow 3G - Messung 1

Vue	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	2,23	2,24	2,24	2,24
	2,24	2,24	2,24	2,24
	2,10	2,10	2,10	2,10
Mittelwert:	2,19	2,19	2,19	2,19

Tabelle A.8.: Messwerte Ladegeschwindigkeit Vue - Slow 3G - Messung 2

React	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	2,24	2,31	2,31	2,31
	2,24	2,31	2,31	2,31
	2,23	2,30	2,30	2,30
Mittelwert:	2,24	2,31	2,31	2,31

Tabelle A.9.: Messwerte Ladegeschwindigkeit React - Slow 3G

Svelte	**DCL in Sekunden**	**FCP in Sekunden**	**TTI in Sekunden**	**LCP in Sekunden**
	2,21	2,21	2,21	2,21
	2,23	2,23	2,23	2,23
	2,24	2,24	2,24	2,24
Mittelwert:	2,23	2,23	2,23	2,23

Tabelle A.10.: Messwerte Ladegeschwindigkeit Svelte - Slow 3G

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 3. November 2022

Vinzenz Herrmann