Hochschule Esslingen
University of Applied Sciences

Pflichtpraktikum

Forschung von leichtgewichtigen Frameworks in der Web Entwicklung

im Studiengang Softwaretechnik und Medieninformatik
der Fakultät Informationstechnik

Konstantinos Tsolakidis
Matrikelnummer: 760311

Zeitraum: 01.03.2021 - 31.08.2021
Prüfer:
Zweitprüfer:

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 14. Juni 2021 _____

Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1 Einführung

in bearbeitung..

1.1 Motivation

in bearbeitung..

1.2 Aufgabenstellung

in bearbeitung..

2 Über IT-Designers

2.1 Gründungsgeschichte

Alles begann als Prof. Goll hat im Jahre 1994 den Grundstein für die IT-Designers Gruppe gesetzt hat. Dabei hat er die Firma als Transferzentrum im Steinbeis-Verbund gegründet. Der Fokus lag dabei den Nachwuchs von IT-Absolventen der Hochschule als auch der eigenen Mitarbeiter zu fördern und das gewonne Wissen an Ihre Kunden weiter zu geben. Im Laufe der Jahre entstand im Jahr 2001 aus dem STZ Softwaretechnik die IT-Designers Gruppe von Prof. Goll und einigen Mitarbeitern. Sie ist heute im Mehrheitsbesitz zweier Stiftungen. Finanziell unabhängig und robust, richtet das Untenehmen die tägliche Arbeit an langfristigem Erfolg, hoher Kundenbindung und überragender Mitarbeiterqualifikation aus. Die IT-Designers, das sind 80 Mitarbeiter, die sich durch einen Abschluss in Informatik oder Mathematik auszeichnen und alle eine mehrjährige Erfahrung in Softwareprojekten in der Industrie nachweisen.

2.2 Thema meines Praktikums

Anfangs hat sich die Arbeit damit befasst, sich mit der Analyse verschiedener Clientseitiger Leichtgewichtiger Web Frameworks auseinander zu setzten. Dafür werden Demo-Applikationen erstellt und miteinander verglichen. Das Thema Leichtgewichtigkeit ist immer von sehr wichtiger Bedeutung, da die meisten Geräte, die für das Surfen im Internet mobile Endgeräte wie Smartphones oder Tablets sind. Im diesem Fall werden Web-Applikationen für Mikrocontroller erstellt. Dabei muss beachtet werden, dass der ROM (Read Only Memory) - Speicher auf einem Mikrocontroller sehr klein ist. Typische Speichergrößen sind 64KB bis 128KB. Die Erstellung der Demo wird dabei evaluiert und anschließend Dokumentiert. Im laufe der Arbeit hat sich auch eine weitere Interessante Aufgabe Herauskristallisiert. Dabei handelt es sich um eine Serverseitige Anwendung. Diese soll eine eigene REST-Schnittstelle sein und die Web-Applikation damit ausliefern.

3 Grundlagen der Webentwicklung

3.1 HTML

HTML ist im world-wide-web grundlegend und essentiell. HTML ausgeschrieben Hypertext-Markup-Language ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente. HTML-Dokumente werden von Webbrowsern dargestellt. In der Webentwicklung gibt es immer eine Root-HTML Seite die es in jedem Projekt geben muss. Sie heißt in den meisten Fällen index.html und an dieses Dokument werden alle anderen Dokumente wie die CSS und JavaScript verknüpft.

3.1.1 Aufbau einer reinen HTML-Seite

Der Aufbau einer HTML-Seite besitzt immer einen HTML-Head. Der Head wird nicht angezeigt und verfügt hauptsächlich technische oder dokumentarische Informationen. Der HTML-Body ist für die Anzeige des Inhalts einer Seite verantwortlich. In ihm befinden sich meistens ein Header, die Main und der Footer.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Titel der Webseite</title>
5      <!-- weitere Kopfinformationen -->
6      <!-- Kommentare werden im Browser nicht angezeigt. -->
7    </head>
8    <body>
9      <p>Inhalt der Webseite</p>
10   </body>
11 </html>
```

Listing 1: HTML-Struktur

3.1.2 HTML-Elemente und ihre Funktionen

in bearbeitung..

3.1.3 HTML-Head

in bearbeitung..

3.1.4 HTML-Body

in bearbeitung..

3.2 CSS

CSS steht für Cascading Style Sheet. Es beschreibt das Aussehen von HTML-Elementen. Dabei ist es ein einzelnes Dokument in dem die Styles verpackt werden, dieses Dokument endet mit css und meistens

nennt man dieses styles.css oder index.css. Damit ein Element gestyled werden kann muss es über die Selektoren angesprochen werden. Neben den Selektoren für zum Beispiel ein Div-Element, gehören auch andere HTML-Elemente, die man direkt mit dem Namen ansprechen kann. Allerdings gelten die Styles dann für alle sich im HTML-Dokument befindlichen gleichnamigen Elementen. Wenn einzelne Elemente angesprochen werden sollen, verwendet man eine ID. Wenn man eine Gruppe von Elementen stylen möchte dann muss man den Klassen-Selektor verwenden. Die Klasse wird mit einem Punkt und den vergebenen Klassennamen („ .klassenName “) im CSS Dokument angesprochen. Außerdem folgt noch ein Block aus geschweiften Klammern in dem die Styles definiert werden. Für die ID das gleiche nur mit einem Hashtag „#“. weiteres in bearbeitung..

3.3 JavaScript

JavaScript ist die populärste Skriptsprache. Sie ist für das Web entwickelt und einfach zu erlernen. Dabei können Inhalte dynamisch verändert, nachgeladen und generiert werden. Anfangs war JavaScript nur für den Browser bestimmt, allerdings hat sich das verändert und es kann auf Servern oder auch Mikrocontrollern gefunden werden. Dabei kann die Skriptsprache

3.3.1 Datentypen

in bearbeitung..

3.3.2 Kontrollstrukturen

in bearbeitung..

3.3.3 Variablen und Konstanten

in bearbeitung..

3.3.4 Funktionen

in bearbeitung..

3.3.5 Objekte

in bearbeitung..

3.4 HTTP

Das Hypertext Transfer Protokol..

in bearbeitung..

3.4.1 Funktionsweise

in bearbeitung..

3.4.2 Anfragemethoden

in bearbeitung..

3.4.3 Statuscodes

in bearbeitung..

3.5 Representational State Transfer (REST)

in bearbeitung..

3.5.1 Prinzipien

in bearbeitung..

3.5.2 Umsetzung

in bearbeitung..

3.5.3 Sicherheit

in bearbeitung..

4 Was ist ein Frontend Framework und was macht es zum Leichtgewicht?

Ein Framework (englisch für „Rahmenstruktur“, „Ordnungsrahmen“) ist kein autonomes Programm, sondern vielmehr eine spezielle Form einer Klassenbibliothek. Ein Framework stellt die Software-Architektur (also das Grundgerüst) einer Anwendung dar und bestimmt wesentlich den Entwicklungsprozess. Frameworks besitzen bestimmte Entwurfsmuster (Design Patterns) mit verschiedenen Funktionen (häufig in Form mehrerer Bibliotheken) und dienen der Entwicklung neuer, eigenständiger Anwendungen.

(Hinzufügen: Erklärung von FrontendWebFrameworks, Visuelle Framworks (CSS))

4.1 Leichtgewichtiges Framework

Jetzt gilt es zu definieren welche Eigenschaften ein Framework zum Leichtgewicht machen. Ein Framework ist in erster Linie Leichtgewichtig wenn die Größe von Anfang an klein ist. Manche Frameworks scheinen Anfangs klein zu sein, doch dies kann sich schnell ändern wenn dadurch nicht alle notwendigen Funktionalitäten inkludiert sind. Andere wiederum sind schon recht klein aber können sogar noch kleiner werden, da sie Modular sind. Modular bedeutet, dass mehrere Module zusammen ein großen ganzes ergeben, diese aber auch alleine isoliert voneinander funktionieren. Modulare Frameworks tauchen Hauptsächlich in CSS-Frameworks auf.

4.2 Welche Frameworks werden getestet?

clientseitiges JavaScript-Framework:

- Mithril
- Preact

Für CSS:

- Purecss.io
- Bulma.io

Datenbanksystem:

- ArangoDB

Serverseitiges microservice-Framework:

- Foxx Microservices für ArangoDB

5 Demo

Eine Demo ist ein zentraler Bestandteil um Funktionen und Vorteile einer Software schnell zu verstehen und testen zu können. Die Erstellung einer Demo unter gleichen Bedingungen mit verschiedenen Frameworks, erlaubt es diese richtig evaluieren zu können.

5.1 Idee

Es soll getestet werden, ob Webapplikationen erstellt werden kann die flüssig auf eingebetteten Geräte laufen kann. Die Webapplikation soll mithilfe einer REST-Anfrage, Daten aus einer Schnittstelle beziehen um daraus eine Bedienoberfläche zu erstellen, die es für den Benutzer möglich macht mit den Daten zu arbeiten. Dabei soll auch der Visuelle Aspekt der Übersichtlichkeit vorhanden sein. Hinzu kommt das diese auch leicht verständlich und bedienbar sein soll.

5.2 Aufbau

Für die Erstellung braucht es im wesentlichen eine Entwicklungsumgebung und Schnittstelle um auf Daten zugreifen zu können. Für die Demo wird eine öffentliche API verwendet die Corona-Daten liefert. Für die Architektur wurde die Model-View-Controller-Architektur verwendet. Diese ist für einen flexiblen Programmentwurf bestens geeignet. Dadurch kann man später Änderungen und Erweiterungen leicht einbauen und die Komplexität nimmt ab. Um nun eine Demo mit Frameworks zu erstellen braucht man zuerst das Grundwissen in HTML, CSS und Javascript. Als erster Schritt wird Recherche betrieben. Dafür eignet sich die Dokumentation der jeweiligen Frameworks am besten. Diese können leicht im Internet gefunden werden. In der Dokumentation gibt es einige Beispiele die ausprobiert werden können. Nach der Recherche baut man nun die Hauptseite auf. Auf der Hauptseite soll es eine Struktur geben. Diese soll einen Header mit der Navigation haben. Einen Body mit der Main, d.h. mit dem Inhalt der Seite der sich immer verändert. Und es soll einen Footer geben. In den Header wird zunächst eine Navigationsleiste eingebaut aber zunächst ohne Logik. Anschließend wird die Main aufgebaut. Da die Main aber die Daten der API-Schnittstelle anzeigen soll, muss zunächst eine verbindung aufgebaut werden. Diese ist mit eine Fetch-Anweisung zu erreichen. Dabei wird eine HTTP-Get Anfrage an eine Schnittstelle gestellt. Die Schnittstelle ist eine URL. Nach der erfolgreichen Fetch-Anweisung werden die damit gewonnenen Daten, visuell auf der Main angezeigt. Dies erfordert fundierte Kenntnisse in HTML und Javascript. Sobald die erste Seite erstellt wurde, beginnt man (falls nicht schon von Anfang an) mit der Trennung des Models und der View. Dabei beschreibt das Model die Form der Daten und die View die Form der Visuellen Anzeige der Daten auf der Webseite. Als nächstes wird das Rounting der erstellten Komponenten eingebunden. Sodass nur die Main verändert wird. Dabei entsteht die Illusion das immer neuer Seiten angezeigt werden. Es handelt sich aber um eine Single-Page-Applikation die nur eine Seite besitzt. Zu guter letzt ist das der Feinschliff mit CSS gefragt. Es soll für mobile genau wie für Desktop - Geräte verwendet werden können.

5.3 Bewertungskriterien

Die Bewertungskriterien sind wie folgt aufgebaut.
Es wird am Ende der Demo-Anwendung, auf

- die Größe der Datei geschaut
- die Komplexität des Codes
- das Bedienbarkeit
- die Schnelligkeit
- den Lern- und Entwicklungsaufwand

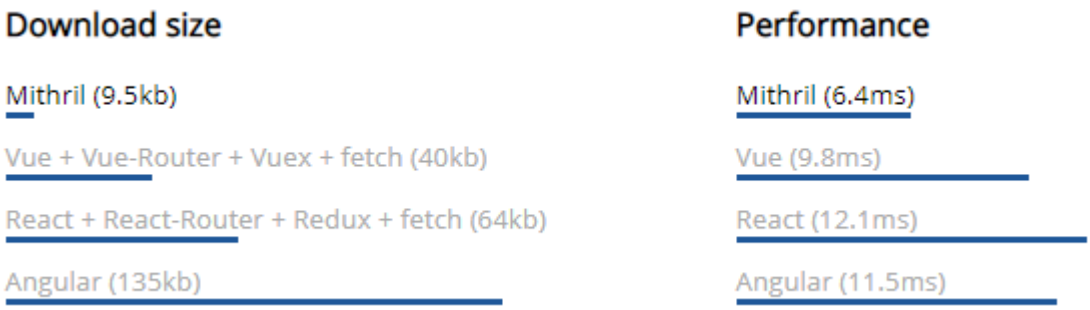

Abbildung 1: Temp Caption

6 Untersuchung von JavaScript Frontend Frameworks

6.1 Mithril

Mithril ist ein modernes clientseitiges JavaScript-Framework zum Erstellen von Einzelseitenanwendungen sogenannten Single Page Application. Es ist klein (¡10 KB gzip), schnell und bietet sofort einsatzbereite Routing- und XHR-Dienstprogramme.

6.1.1 Installation

Die installation kann auf verschiedene Wege realisiert werden.

Wenn es einen Node.js server gibt, dann kann Mithril mithilfe des Node-Package-Managers installiert werden. Dafür muss nur der folgende Befehl im Terminal eingegeben werden, wenn man sich im richtigen Ordner befindet.

- npm install mithril

Ein anderer Weg ist über das Content Delivery Network (CDN). Dafür muss der Link einmalig in Body der root HTML-Datei eingebunden werden. Im normalfall handelt es sich hier um die index.html.

```
1  <script src="https://unpkg.com/mithril/mithril.js"></script>
```

Listing 2: CDN-Link für Mithril

Auch ein sehr intressanter Weg das Framework zu nutzen, ist es lokal einzubinden. Dafür muss eine Javascript-Datei erstellt werden, die die Distribution des Frameworks beinhaltet. Die Distribution kann mithilfe von unpk gefunden werden oder auf dem Github repository des Frameworks.

6.1.2 Wichtige Mithril Funktionen

Die m-Funktion ist eine Hyperscript-Funktion die es erlaubt jedes DOM-Element zu erstellen, das HTML besitzt. Zum Beispiel kann ein div damit erstellt werden. Das dieses div aber mit Mithril erstellt wurde, ist kein echtes DOM sondern ein Virtual-DOM oder auch vnode genannt. Das vnode ist ein JavaScript Objekt was an ein echtes DOM angehängt wird. Es reagiert auf veränderung deutlich

schneller als ein echtes. Das DOM-Element an das die vnodes angehängt werden ist dabei fast immer der Body. Die Schreibweise ist sehr Kompakt und kann sehr übersichtlich gestaltet werden.

Die Funktion nimmt drei Argumente getrennt mit einem Komma an. Das erste Argument ist das HTML-Element was erstellt werden soll. Wenn dieses Feld leer bleibt, handelt es sich um ein div. Das zweite Argument sind die Attribute die es haben kann. Diese sind optional. Die Attribute werden in geschweiften Klammern geschrieben. Darin können sich styles befinden. Das letzte argument ist das Child. Das der Wert zwischen den HTML-tag. Also das was auf der Ausgabe in der Webseite angezeigt werden soll. Außerdem kann man noch eine kürzere schreibweise für die Klassen und ID Selektoren für CSS verwenden, indem man diese mit . für Klasse und # für ID in das erste Argument schreibt. Im folgenden Beispiel wird dies nochmal deutlich gemacht.

```
1  m("div.klasse", \{id: "box"\}, "hello")
2  //Equivalent in HTML:
3  //<div id="box" class="klasse"> Hallo </div>
```

Listing 3: Beispiel einer konfigurierten m().

In folgendem Beispiel wird noch das zweite Argument näher erklärt. Dabei werden dem button drei Attribute übergeben. Alle Attribute müssen in die geschweifte Klammer und werden mit Kommas voneinander getrennt.

```
1  m("button", {
2    class: "my-button",
3    onclick: function() {/* ... */},
4    oncreate: function() {/* ... */}
5  })
```

Listing 4: Einbinden von Klassen, Eventhandlern und Lifecycle Methoden

6.1.3 Mithril Komponente

Eine Komponente ist ein JavaScript Objekt und sie kann eine View-Funktion beinhalten. Außerdem kann eine Komponente Funktionen enthalten. Lokale Variablen können in eine solche Komponente erstellt werden. Der zugriff auf diese Variablen ist über das Ansprechen der Komponente und die darin befindlichen variablen mit dem Punkt-Operator möglich. Die Komponente kann auch komplett mit der Model-View-Controller Architektur erstellt werden. Um Daten in eine Komponente zu übertragen muss das über das attrs-Objekt von Mithril erfolgen. Ein Beispiel um dieses vorgehen zu verstehen:

Um zugriff auf die Komponenten zu bekommen muss die Komponente exportiert und importiert werden, wie in folgenden Beispielen aufgezeigt wird.

```
1  import BeispielKomponente from "./BeispielKomponente.js"
2  const beispielVariable = BeispielKomponente.beispielData;
```

Listing 5: Zugriff auf Daten einer importierten Komponente.

```
1  let BeispielKomponente : {
2    beispielData: "Hallo",
3  }
4  export default BeispielKomponente;
```

Listing 6: Exportieren einer Komponente.

6.1.4 Mithril REST-Anfragen mit m.request

Mithril liefert eine Funktion mit der man Anfragen schicken kann. Die sogenannte m.request(). Die Funktion ähnelt sehr der JavaScript Fetch-API. Als Response bekommt man immer ein Promise zurück. Egal ob die Anfrage Problemlos funktioniert hat oder nicht.

```
Promise {<pending>, constructor: ?, then: ?}
constructor: ? PromiseProxy(executor)
then: ? ()
__proto__: Promise
catch: ? catch()
constructor: ? Promise()
finally: ? finally()
then: ? then()
Symbol(Symbol.toStringTag): "Promise"
__proto__: Object
[[PromiseState]]: "fulfilled"
[[PromiseResult]]: undefined
```

Listing 7: Promise einer REST-Anfrage mit m.request().

Die Anfrage braucht mindestens eine definierte HTTP-Methode und die URL von der aus die Daten angefragt oder hinterlegt werden sollen. Je nachdem was es für eine Methode ist muss auf weiteres geachtet werden. Für die GET-Anfrage ist es möglich in der URL einen Suchbegriff oder ähnliches einzufügen. Dafür einfach über String-Verkettung mit einem + den erwarteten input weiterleiten.

```

let getRequestWithUserInput = {

  fetchExample: function (usereingabe) {
    return m.request({
      method: "GET",
      url: "URL" + usereingabe,
    })
  }
```

Listing 8: Beispiel: Request der Methode GET mit Benutzereingabe.

Für weitere Möglichkeiten wie die PUT oder Post können auf der Dokumentationsseite von Mithril.js recherchiert werden.

Um nun mit den Daten der Antwort arbeiten zu können, kann wie im nächsten Beispiel einen .then / .catch Block angewendet werden.

```

let fetchExample = {
  response: [],

  fetchEx: function (input) {
    return m.request({
      method: "GET",
      url: "URL" + input,
    }).then(function (data) {
      try {
        fetchExample.response = data;
      } catch (error) {
        console.log("Error:" + error);
```

```
14          }
15        });
16      }
```

Listing 9: Beispiel: m.request().then().catch().

6.1.5 Lebenszyklen in Mithril

Komponenten können über eine oder mehrere Lebenszyklus Methode verfügen, die an verschiedenen Punkten während der Lebensdauer eines DOM-Elements aufgerufen werden kann. Die von Mithril unterstützten Lebenszyklus Methoden sind: oninit, oncreate, onupdate, onbeforeremove, onremove und onbeforeupdate.

Die Lebenszyklen werden spätestens dann wichtig wenn eine Fetch-Anweisung Daten von einer Schnittstelle laden soll. Die dabei unterschiedlich lange dauern kann. Dadurch ist nicht automatisch gewährleistet ob eine komponente zugriff auf die Daten hat, wenn diese erstellt wurde. Für so einen Fall empfielt sich die Mehtode oninit. Dabei werden die Daten als erstes geladent und erst wenn diese Vollständig da sind wird die Komponente gebaut.

6.1.6 Mithril Routen

Das Routen oder Navigieren der Seiten in einer Anwendung ist ein essentieller Bestandteil jeder Webapplikation und Webseite. Diese ist in Mithril wie folgt gelöst. Mithril verwendet dafür eine eigene Mehtode. Die methode wird m.route genannt und wie in folgendem Beispiel angewendent. Dabei wird einfach eine Komponente eingebaut die angezeigt werden soll. Das erreicht man zum Beispiel durch eine Statusvariable.

Der Aufbau sieht wie folgt aus:

```
1     m.route(root, defaultRoute, routes)
```

Listing 10: Aufbau der m.route-Funktion.

Die Funktion verlangt 3 Parameter. Der erste ist das DOM-Elemnt an das alles angehängt wird. Wie zum Beispiel das Body Element das über document.body erreichbar ist. Der zweite Parameter ist eine Default Route damit eine Startseite angezeigt werden kann. Der dritte Parameter ist ein Objekt mit weiteren Routen.

Ein Beispiel aus der Demo sieht wie folgt aus:

```
1
2  m.route(document.body, "/home", {
3    "/home": {
4      render: function () {
5        return m(home);
6      },
7    },
8    "/country": {
9      render: function () {
10       return m(country);
11     },
12   },
13   "/vaccine": {
```

```
14      render: function () {
15        return m(vaccine);
16      },
17    },
18    "/allCountries": {
19      render: function () {
20        return m(allCountries);
21      },
22    },
23  });
```

Listing 11: Aufbau der m.route-Funktion.

Wie in dem Beispiel zu sehen, wird immer die entsprechende Komponente, die auch die View beinhaltet übergeben. Um die Routen in der Navigationsleiste richtig aufzurufen baut man einen Link ein der im href die route enthält. Eine mögliche implementierung in der Navigationsleiste könnte wie folgt aussehen:

```
1  m(
2  "a[href=./index.html#!home]",
3    {
4    /* Mit m.route.link wird der angegebene Link in href übergeben. */
5    oncreate: m.route.link,
6    },
7    "HOME"
8  ),
```

Listing 12: Aufbau der m.route-Funktion.

6.1.7 Dynamische Inhalte in Mithril

Dynamische Inhalte d.h. das verändern von Inhalten, oder das abrufen von Daten zur Laufzeit ist bei Beachtung der bisher beschriebenen Punkte möglich. Sobald eine Variable verändert wird, oder eine Fetch-Anweisung getriggert wird, erzwingt man damit ein neues zeichnen der jeweiligen Komponente auf der Webseite.

6.1.8 Fazit

Mithril bietet im gesamten eine breites Verzeichnis für die Erstellung von kleinen Webanwendungen an. Eine Komponente ist dabei übersichtlich in drei Teile aufgeteilt. Für Übersichtlichkeit sorgt auch die Struktur, die mit der Hyper-Script-Funktion von Mithril erreicht werden kann. Auch das auslagern von Teilkomponenten ist leicht und übersichtlich realisierbar. Die Dokumentation ist sehr Umfangreich, allerdings ist das Framework dennoch nicht leicht zu erlernen und hebt sich durch die Struktur von anderen Frameworks ab. Es ist auch nicht weit verbreitet und es hat eine vergleichsweise kleine Gemeinschaft, dadurch können Fehler erschwerter behoben werden.

6.2 Preact

6.2.1 Installation

Preact hat eine dist.js Datei. Diese kann man entweder über einen CDN-Link einbinden, oder man kann sich diese Distribution kopieren und lokal in das Projekt als eigene Bibliothek einbinden. Wir haben uns dieses mal für eine lokale Bibliothek entschieden. Deshalb habe ich die Distribution über den Github Kanal heruntergeladen und lokal in das Projekt eingebunden. Allerdings muss man darauf achten ob diese auch alle Abhängigkeiten abdeckt. Gegenbefalls müsste man weitere miteinander Verknüpfen.

```
1  import { h, Component } from './PfadZurBib.js'
```

Listing 13: Import der Funktionen/ Mehtoden in Preact über eine Lokale Distribution

6.2.2 Struktur

Die Struktur von Preact ist sehr ähnlich der von React. Es ist von Vorteil wenn man sich schon mit React auskennt. Die Dokumentation in Preact ist nicht so ausführlich weil man vieles aus der Dokumentation von React bekommt. Es gibt dennoch signifikante unterschiede. Ein großer unterschied ist das Preact kein JSX unterstützt und man durch eine zusätzliche Bibliothek mit dem Namen „htm“ einbinden muss. htm wandelt den JSX code in normalem HTML Code um, sodass man nur jede Variable oder Komponente mit einem „${Variable}“ angeben muss. Die ganze HTML Struktur wird nun in Javascript erstellt.

In folgender Abbildung ist der Aufbau einer Funktionalen Komponente die, die HTML-Struktur zurück gibt. Wie man auch sehen kann ist die Naviagtions-Komponente wie oben erwähnt eingebunden. Und darunter sieht man die gewohnte HTML-Schreibweise.

Was hier aber erstellt wird ist kein echtes HTML-Objekt. Es handelt sich hierbei um ein Virtual-DOM.

```
1  import { h, render } from "./libs/distPreact.js";
2  import htm from "./libs/distPreactHTM.js";
3  const html = htm.bind(h);
4  import { Navigation } from "./Components/navigation.js";
5
6  const Home = () => {
7    return html`
8    <\${Navigation} />
9    <footer class="footer mt-6">
10     <div class="content has-text-centered">
11       <p>
12         <strong>Covid Demo mit Preact.js and Bulma.io</strong> erstellt von
               Konstantinos Tsolakidis
13       </>
14     </div>
15   </footer>
16   `;
17 };
18 render(h(Home), document.body);
```

Listing 14: Aufbau einer HTML-Struktur mit Preact

6.2.3 Klassen-Komponente

Eine Klassen Komponente erbt von der preact-Methode Component. Sie braucht außerdem einen Konstruktor. In diesen Konstruktor werden dann die States d.h. die Statusvariablen der Klassenkomponente erstellt. Außerdem ist man gezwungen an jede Funktion in dieser Klasse das this an jede Funktion im Konstruktor anzubinden. Sie enthält außerdem noch eine Mehthode rendern() die das HTML zurückgibt. Es wird in diesem Fall mit htm als HTML geschrieben und nicht wie in React in JSX. Mit einer Veränderung der States wird ein neu rendern ausgelöst. Hinzu kommt das man sich jetzt auch mit den Lebenszyklen befassen muss. Denn es gibt einige dieser Methoden, aber eine dieser Methoden ist sehr wichtig. Die Methode ComponentDidMount() wird dazu genutzt um Datenzugiff einer Fetchanweisung zu garantieren, denn diese sorgt dafür das wenn die HTML Struktur aufgebaut wurde auch der Zugriff auf die Dateien erfolgt.

6.2.4 Funktionale-Komponenten

Eine Funktionale Komponente wird mit der neuen Schreibweise von ES6 erstellt. Das heißt es werden Pfeil Funktionen erstellt. Der große Vorteil hierbei ist die kurze Schreibweise und der Entfall des this.

Um in einer Funktionales Komponente mit States zu arbeiten wird die Hauseigene Methode der Hooks von Preact angewendet. Dafür muss die Hooks Bibliothek importiert werden. Diese beinhaltet dann zwei Methoden. Eine ist die useState() und die zweite ist die useEffect(). Die useState wird für die States verwendet. Dabei wird in einer Zeile der Typ des States in den Klammer des useStates() geschrieben und auf der anderen Seite ein Setter und die eigentliche Variable die den Wert beinhaltet.

Die useEffect() ist eine Lebenszyklus-Methode die genau wie die ComponentDidMount() funktioniert. Dabei werden alle Funktionen die in dieser Methode aufgerufen werden garantiert nach der Erstellung des DOM-Baums zur Verfügung stehen.

6.2.5 Lebenszyklen

Lifecycle method When it gets called componentWillMount before the component gets mounted to the DOM componentDidMount after the component gets mounted to the DOM componentWillUnmount prior to removal from the DOM componentWillReceiveProps before new props get accepted shouldComponentUpdate before render(). Return false to skip render componentWillUpdate before render() componentDidUpdate after render()

6.2.6 HTTP-REST Anfrage mit Preact

Preact besitzt keine eigene Fetch-Methode. Es wurde die Javascript Fetch-API benutzt.

6.2.7 Preact States und Props

States werden über den Konstruktor in einer Klassenkomponente oder in der useState() in einer Funktionalen Komponente erstellt und benutzt. Die Props sind dazu da um Daten in Child-Komponenten zu versenden. Dafür werden die daten mit einer Beliebigen Variable in den HTML-Code der Komponente geschrieben. In folgendem Beispiel bekommt die Komponente Modal einen Prop übergeben. Dieser Prop kann dann in der Komponente Modal über prop.data aufgerufen und benutzt werden.

```

    <${Modal} data=${currentCountry} active=${active} />
```

Listing 15: Preact Props einer Komponente übergeben.

6.2.8 Preact Routing

Für das Routing wird eine Bibliothek mit dem namen Preact-Router genutzt und noch eine mit dem Namen history.

Wie im folgenden Beispiel zu sehen werden in der Router Komponente die Pfade gesetzt. Damit diese auch besser gefunden werden und auch die zurück taste im Browser verwendet werden kann, wird der gebrauch von createHistory() verwendet.

```

<${Router} history=${createHashHistory()} >
  <${HomePage} path="/" />
  <${Dashboard} path="/dashboard"/>
  <${Vaccine} path="/vaccine"/>
  <${Country} path="/country"/>
</$>
```

Listing 16: Preact Router mit history.

In folgendem Codebeispiel sieht man die Einbindung einer Route in die Navigationleiste. Hierfür wird ein Link-Tag benutzt und mit der Route und einem onClick-Handler versehen.

```
<a class="navbar-item" href="/country" onClick=${()=> setActive("")}>
```

Listing 17: Route in einem Link eingebunden.

6.2.9 Dynamische Inhalte in Preact

Dynamische Inhalte sind in Preact einfach zu gestalten. Dafür muss mit den States gearbeitet werden. Diese führen einen neues rendern aus. Die Veränderung der States werden sofort auf der Seite angezeigt.

6.2.10 Fazit

Preact ist eine abgespeckte Version von React. Dabei kann es und muss es teilweise erweitert werden. Preact braucht die Bibliothek „HTM“. Auch die Benutzung von Bibliotheken die für React gedacht sind, sind teilweise kompatibel mit Preact. Es ist deutlich kleiner (3KB) als React (45KB) und auch von der Performance schneller, dies macht es für mobile Geräte sehr Interessant. Dabei ist die größte Einsparungen, dass weglassen von einem Hauseigenen synthetischen Event System. Preact verlässt sich auf den Haus eigenen Eventlistener der vom Browser zur Verfügung gestellt wird. Allerdings ist das nur ein Teil der Funktionen von React darunter fehlen noch PropTyps und children. Ein weiterer Punkt der auch zu beachten ist, ist das Preact eine deutlich kleinere aktive Gemeinschaft hat. Das macht es schwieriger bei Problemen oder Fehlern den Grund aufzufinden. Die Dokumentation ist recht klein, da auch manches in der Dokumentation von React nachgeschlagen werden kann. Die Struktur ist sehr sauber und das arbeiten mit Funktionalen Komponenten und Hooks spart sehr viele Zeilen Code und erleichtert dabei auch mit dem Verständnis. Auch ist durch den Einsatz von „HTM“ eine nahezu identische Erstellung einer „normalen“-HTML Struktur möglich. Dadurch spart man sich viel Zeit beim erlernen des Frameworks.

Abbildung 2

7 Untersuchung von CSS-Frameworks

7.1 Purecss

Purecss ist im gesamten ein 3.7 KB Großes Framework. Es ist Modular aufgebaut. Darunter befinden sich insgesamt 6 einzeln benutzbare Module. Diese sind unabhängig voneinander Installierbar und können somit deutlich weniger Speicherplatz einnehmen. Da die Module sehr klein sind, haben diese deswegen auch keine große Auswahl. Aber für kleine Projekte reichen die Module aus. In der Abbildung 2 ist die Modulare Aufteilung mit ungefährer Speichergröße abgebildet. Unter folgendem Link: https://purecss.io/customize/ stehen die Links für jedes einzelne Modul.

7.1.1 Installation / Einbindung in das Projekt

Man kann das Framework über CDN (Content Delivery Network) einbinden. Dafür muss lediglich der folgende Link in den Head der HTML Datei eingefügt werden.

```
1
2 <link rel="stylesheet" href="https://unpkg.com/purecss@2.0.5/build/pure-min.css"
    integrity="sha384-G9DpmGxRIF6tpgbrkZVcZDeIomEU22LgTguPAI739bbKytjPE/
    kHTK5YxjJAAEXC" crossorigin="anonymous">
```

Listing 18: Einbinden von purecss.io

7.1.2 Pure Grid Layout

Das Gridlayout von purecss ist einfach aufgebaut und lässt sich komplett individualisieren. Die Grids lassen sich in zwei Gruppen aufteilen. Dabei hat es die Einteilung in grob und fein. Dabei kann beim groben layout eine Einteilung von 5 Spalten erreicht werden. Bei der feinen sogar 24. Um ein solches Layout zu realisieren, muss als erstes ein Div erstellt werden, dass als Container dient. Dieses bekommt die Klasse pure-g. In diesen kommen dann die einzelnen Elemente mit der Klasse pure-u-*. Dabei steht das Sternchen für die jeweilige Breite, die erreicht werden soll. Diese kann zum Beispiel von der ganzen Breite bis hin zu einem vierundzwanzigstel klein sein. Des weiteren lassen sich auch Bilder in die einzelnen Spalten integrieren. Dafür muss ein img-Tag in das div mit der Klasse pure-img integriert werden. Dabei wird das Bild responsive und lässt sich dynamisch skalieren.

In nächsten Beispiel wird gezeigt, wie die sich der Klassenname zusammenstellt, wenn man diese wie in css mit der Media Query anpassen möchte. Dabei wird in jeder Klasse der entsprechende Key eingebaut. Somit schreibt man für kleine Endgeräte unter 568px ein sm für small und für Geräte größer 1280px ein xl für extra Large.

Das Gridlayout funktioniert nur bei dem div-Element.

```
1
2   Key     CSS Media Query                    Applies        Classname
3   None    None                             Always         .pure-u-*
4   sm    @media screen and (min-width: 35.5em)    ? 568px       .pure-u-sm-*
```

```
  md     @media screen and (min-width: 48em)      ? 768px       .pure-u-md-*
  lg       @media screen and (min-width: 64em)      ? 1024px     .pure-u-lg-*
  xl       @media screen and (min-width: 80em)        ? 1280px     .pure-u-xl-*
```

Listing 19: Aufteilung der @media screen größen in purecss.io

In folgendem Codebeispiel wird veranschaulicht wie das Gridlayout von purecss in Mithril bei einem div eingebunden wird.

Die Größe des Endgerätes bestimmt nun wie viel Platz das Div auf dem Bildschirm bekommt. Auf einem mobilen Endgerät wie dem Smartphone würde dieses Div 100 Prozent des Bildschirm einnehmen. Bei einem Tablet wären das, dann nur noch 50 Prozent und bei einem Desktop PC mit einem Monitor Größer als 1024px nur 25 Prozent der Bildschirmbreite.

```

m("div.pure-u-1 pure-u-md-1-2 pure-u-lg-1-4",{attrs},child)
m("div.pure-u-1 pure-u-md-1-2 pure-u-lg-1-4",{attrs},child)
m("div.pure-u-1 pure-u-md-1-2 pure-u-lg-1-4",{attrs},child)
m("div.pure-u-1 pure-u-md-1-2 pure-u-lg-1-4",{attrs},child)
```

Listing 20: Beispiel: Responsive Grid implementieren in MithrilJs.

7.1.3 Pure Buttons

Das Button Modul ist ein sehr kleines und praktisches Modul. Es beinhaltet keine sehr große Auswahl an verschiedenen Knöpfen. Aber die Styles, die es anbietet sind durchaus brauchbar und können sich trotzdem sehen lassen. Neben den Standard Knöpfen in Grau gibt es auch welche für einen aktiven und inaktiven Knopf. Außerdem bietet Pure auch den Primary Button der oft in Blogs oder Shops seinen Platz findet. Zuzüglich zur Primary Farbe gibt es 4 weitere. Grün, Rot, Orange und Hellblau die alle einen sehr aussagekräftigen aber nicht störenden Farbton haben. Dabei steht Grün für einen Erfolgreichen Knopfdruck, Rot für einen Fehler, Orange für eine Warnung und Hellblau für den Secondary Knopf. Als letzte Eigenschaft kann die Größe verändert werden. Die größen sind dabei von der Schriftgröße abhängig. Die kleinste Größe ist .button-xsmall und das entspricht der Font-Größe von 70% und der Größte Knopf ist der .button-xlarge und entspricht der Font-Größe 125%.

7.1.4 Pure Forms

Es gibt 5 verschiedene Form-Felder die direkt ürbernommen werden können. Dabei gibt es die Standard Form für den Login mit Email, Password, einem check Feld und einem Button. Diese ist Horizontal angeordnet und für jedes Beispiel ist der Aufbaus in einer HTML-Struktur abgebildet. Die Felder lassen sich sogar in einem Gridlayout verbinden sodass man eine Form bekommt die für Kundendaten geeignet ist.

Die Demo hat für das Suchfeld die Form mit den abgerundeten Ecken bekommen.

```
<form class="pure-form">
  <input type="text" class="pure-input-rounded" />
  <button type="submit" class="pure-button">Search</button>
</form>
```

Listing 21: Verwendete Pure Form in der Demo

7.1.5 Fazit

Pure ist ein sehr kleinen Framework was das minimalste an Styles mit sich bringt. Für kleinere Webanwendungen die keine großen extras wie Karten oder Modals brauchen, eignet es sich gut. Es ist sehr klein und kann durch seine Modularität noch kleiner werden, da auch nur die erwünschten Module installiert werden können. Die Dokumentation ist sehr Gut und die Handhabung sehr einfach.

7.2 Bulma

Bulma ist ein open source Framework das ohne JavaScirpt geliefert wird. Des weiteren ist es Modular. Das bedeutet, dass man auch einzelne Module installieren/ downloaden kann. Deshalb ist es für alle Projektgrößen geeignet. Derzeit ist es in sechs Module unterteilt. Darunter befinden sich die Module Columns, Elements, Components, Forms, Layout und Helpers.

7.2.1 Installation

Die Installation erfolgt entweder über CDN oder über npm (Node Package Manager). Unter folgendem Link: https://bulma.io/documentation/overview/start/ sind die Schritte für die Installation beschrieben. Mit dem Link: https://bulma.io/documentation/overview/modular/ sind die Schritte für die Modulare Installation des Frameworks beschrieben.

7.2.2 Modul Elemente

Das Elements Modul ist eines der essentiellen. Darin befinden sich 12 Elemente wie Block, Box, Button, Content, Delete, Icon, Image, Notification, Progress bars, Table, Tag und Title. In der Demo wurde von Block, Button, Content und Title gebrauch gemacht. Das Element Block wird dafür verwendet einen Abstand zwischen anderen Elementen zu schaffen. Das Element Title wird zusammen mit Subtitle verwendet. Beide haben 6 verschiedene Größen. Wenn Title und Subtitle direkt untereinander benutzt werden rücken diese etwas näher zusammen. Das Content Element wird nur für Text verwendet. Darin dürfen sich nur html-Elemente wie paragraphs, lists, headings, quotes und tables befinden.

7.2.3 Modul Layout

Im Modul Layout befinden sich Styling Komponenten für die Formatierung von Texten und Abschnitten. Für die Demo wurde die Klasse Container und Section verwendet. Um das Styling in das Projekt zu bekommen muss lediglich der Name Container an ein div-tag als Klasse übergeben werden. Bei der Klasse Section braucht es ein section.tag.

```
1
2  <div class="container">
3  <section class="section">
4  <h1 class="title">Section</h1>
5  <h2 class="subtitle">
6  A simple container to divide your page into <strong>sections</strong>, like the
       one you're currently reading.
7  </h2>
8  </section>
9  </div>
```

Listing 22: Beispiel: Klassen Anwendungsbeispiel in Bulma

Es ist auch möglich weitere Veränderungen vorzunehmen. Zum Beispiel wie breit der Container sein soll mit .is-widescreen oder .is-fullhd. Der Container wird genutzt um den Inhalt mittig zu fixieren. Somit ist er immer mit und hat einen default Mindestabstand von 32px zur Außenkante des Bildschirms.

7.2.4 Modul Form

Das Form Modul sorgt für eine Klare Struktur und Übersichtlichkeit des Steuerelements Form. In diesem Modul werden folgende HTML-Elemente gestyled:

label, input, textarea, select, checkbox, radio, button und help.

Im Allgemeinen bietet Bulma für die gängigsten Form Anwendungsfälle solide Lösungen. Diese sind Modern und Schlicht gehalten. Können aber auch mit Icons Individualisiert werden. Dabei reichen die vorgefertigten Lösungen von einer normalen Anmeldung mit Email bis zu Feldern die mit Währungen und Online-shops zu tun haben.

7.2.5 Navigation, Karte und Modal mit dem Modul Komponente

Das Modul Komponente besteht aus 10 Elementen. Darunter befinden sich die Komponenten: Breadcrumb, Card, Dropdown, Menu, Message, Modal, Navbar, Pagination und Panel. Die Elemente sind in grob in zwei Gruppen aufzuteilen. Die erste Gruppe ist die, der Navigation. Das heißt für die Darstellung von Hierarchien und das Navigieren zwischen verschiedenen Seiten. Die zewite Gruppe ist für Effekte und Übersicht geeignet. Die Auslagerung von Informationen wie in einem dropdown Menü oder einem Modal ist hier möglich. Auch eine schöne und übersichtliche Darstellung von zusammengehörenden Inhalten kann mit einer Karte realisiert werden. Für die Demo wurde die Navigation, die Karte und das Modal verwendet. Die Navigation ist schon für kleine und große Geräte optimiert. Wenn ein Smartphone verwendet wird wechselt die Navigation in ein Burger-Menü. Bei einem Desktop mit mehr als 1200 Pixel breite wird die Navigation Horizontal oder Vertikal mit allen einzelnen Elementen angezeigt. Das öffnen und schließen der dropdown Eigenschaften wie beim Burger-Menü wird über den Klassennamen is-active verändert. Das kann mit Javascript und dem dazugehörigen Framework in unterschiedlichen Variationen gelöst werden. Das gleiche gilt auch für das Modal, dass auch durch den Klassennamen is-active gesteuert wird.

7.2.6 Individualisierung mit dem Modul Helpers

Jede Komponente kann noch Individualisiert werden. Und das ohne eine Zeile Code in einem CSS Dokument zu schreiben. Punkte wie das Margin zu anderen DOM-Elementen oder auch die Textfarbe lassen sich einfach mit dem Klassennamen m* für margin und p* für padding und die dazugehörige Seite wie t für Top erstellen. Wenn jetzt der Abstand von oben vergrößert werden soll muss lediglich ein mt-(0-6) eingegeben werden. Die values sind von 0 - 6 und mit einem abstand von 0 , 0.25 rem bis 3 rem gestuft. REM ist relativ zur html fontgröße dagegen ist em abhängig von direkt umligenden Elternknoten.

7.2.7 Fazit

Bulma ist ein modernes und Modulares Framework. Es ist responsive und für mobile first designed. Die Dokumentation ist sehr Umfangreich und leicht verständlich. Die Anwendung ist sehr einfach. Jede Komponente hat ein Beispielcode der die HTML-Struktur aufzeigt. Durch die 39 .sass Dateien lassen sich auch nur die Module einbinden die auch gebraucht werden. Deshalb ist die größe des Frameworks individuell einstellbar und anpassbar nach den Bedürfnissen.

8 Untersuchung von ArangoDB

in bearbeitung...

9 Untersuchung von Foxx-microservice Framework

in bearbeitung...

10 Schluss

10.1 Zusammenfassung

10.2 Kritik und Ausblick