HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING – AUSTRIA

Datenbank Verwaltungstool

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Medientechnik

Eingereicht von:
David Altenhofer
Sami Abbas Ali
David Precup

Betreuer:
Michael Palitsch-Infanger

Projektpartner:
Doka GmbH

Leonding, Juli 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

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

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022 David Altenhofer & David Precup

Abstract

The Doka company offers software packages to their customers. Here is the administration still very much in need of improvement. The process of adding new software packages, works as follows: creating new packages, editing and deleting takes place directly via the database, which means that there is no administration program to date. Since everything runs through the database, it becomes clean work prevented. That's why we were commissioned by the Doka company to create a Database Management System based on the JavaScript library React. Before some terms need to be explained:

It should be possible to manage the following tables that have these properties:

- InstallablePackage is the main table, consists of several Installables and can have a description (InstallablaPackagesDesciptions) in multiple languages.
- An Installable has an InstallableSyncTemplate and can have an InstallableExecutablePaths.

The program should have the following functions:

- Viewing, installing, adding, editing and deleting so-called
Installables / InstallablePackages / InstallableSyncTemplates / InstallableExecutablePaths - Exporting the current configuration
- Importing a JSON file, where to see the difference of the imported file and the current configuration
- Reset the cache in the backend

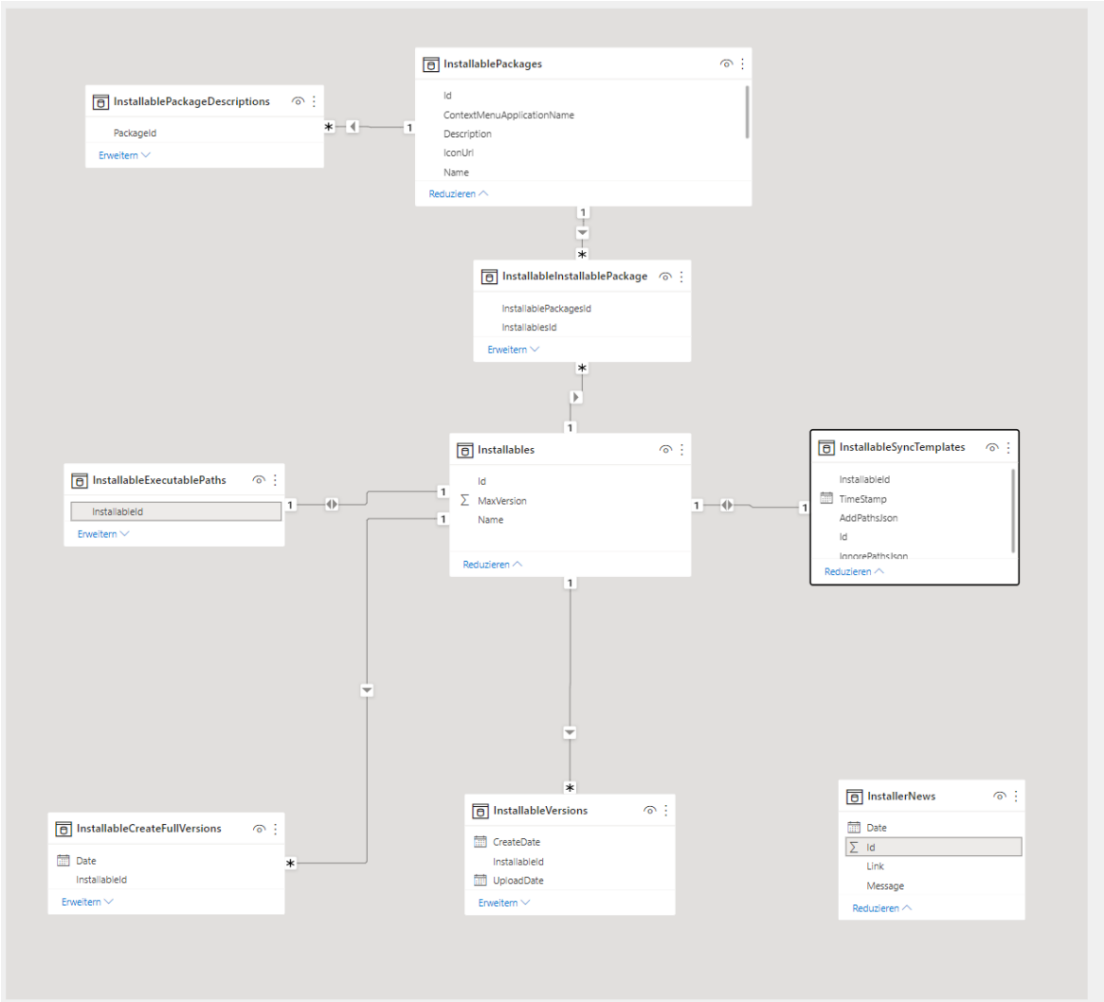

Abbildung 1: ERD

Zusammenfassung

Die Firma Doka bietet Softwarepakete an deren Kunden an. Dabei ist das Verwalten noch sehr verbesserungswürdig. Der Prozess des hinzufügens von neuen Softwarepaketen, läuft wiefolgt ab: Das Erstellen von neuen Paketen, das Editieren sowie das Löschen erfolgt direkt über die Datenbank, das bedeutet, dass es kein Verwaltungsprogramm bis dato gibt. Dadruch, dass alles über die Datenbank abläuft, wird das saubere Arbeiten verhindert. Deshalb wurden wir von der Firma Doka dazu beauftragt, ein Verwaltungssystem basierend auf der JavaScript library React zu programmieren. Davor müssen einige Begriffe noch erklärt werden:

Folgende Tabellen sollen verwaltet werden können, die diese Eigenschaften haben:

- InstallablePackage ist die wichtigste Tabelle, besteht aus mehreren Installables und kann Beschreibung (InstallablaPackagesDesciptions) auf mehreren Sprachen haben.
- Ein Installable hat ein InstallableSyncTemplate und kann ein InstallableExecutablePaths.

Dabei soll das Programm folgende Funktionen haben:

- Das Anzeigen, Installieren, Hinzufügen, Bearbeiten und Löschen von sogenannten Installables / InstallablePackages / InstallableSyncTemplates / InstallableExecutablePaths
- Das Exportieren der aktuellen Konfiguration
- Importieren eines JSON Dateis, wo die Differenz der importierten Datei und der aktuellen Konfiguration zu sehen ist
- Reseten des Cache im Backend

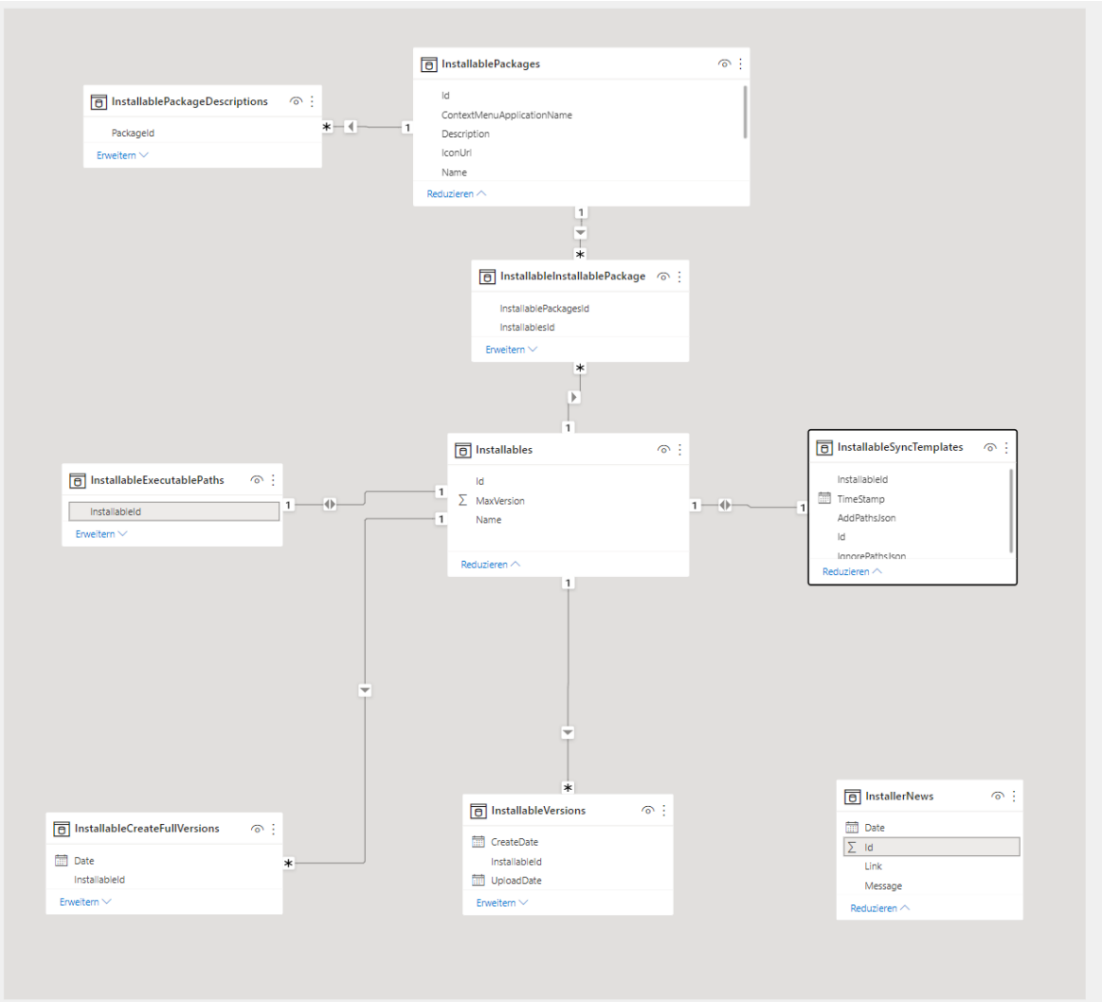

Abbildung 2: ERD

Autoren der Diplomarbeit

David Altenhofer

Aufgabenbereich Frontend und Authentifizierungslogik

Name: David Altenhofer

Geburtsdatum: 8. Juli 2003

E-Mail: david.altenhofer@gmail.com

Bildungsweg

2009 bis 2010 Vorschule Karlhof

2010 bis 2014 Volkschule Karlhof

2014 bis 2018 Europagymnasium Auhof (Kepler-Zweig)

Seit September 2018 in der HTL-Leonding (Medientechnik)

Berufliche Erfahrung:

Sommer 2023 Doka GmbH, Frontend Entwickler

Sprachliche Kenntnisse:

Deutsch Muttersprache

Englisch

Danksagung

An dieser Stelle möchten wir uns bei denen bedanken, die uns bei der Durchfuhrung dieses Projektes beraten und unterstützt haben. Besonderer Dank, gilt Alex und Mathias, die als Mitarbeiter unserer Partnerfirma Doka, uns im Laufe des Projekts fachlich geholfen haben, bei der Planung als auch bei der Durchführung und Lösung von Hindernissen. Ein weiteres Dankeschön an die Firma Doka, die uns unser Diplomarbeitsprojekt im Rahmen eines Praktikums im Sommer zur Verfügung gestellt haben.

Zu guter letzt möchten wir uns noch bei unserem Betreuer Herrn Prof Palitsch-Infanger, für die Beratung und Begleitung dieser Diplomarbeit, bedanken.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Ausgangssituation 1

1.2 Aufgabenstellung 4

1.3 Zielsetzung 4

2 Technologien **5**

2.1 Visual Studio Code 2023 5

2.2 Latex 6

2.3 HTML 8

2.4 CSS 9

2.5 Tailwind CSS 10

2.6 React 18.2.0 12

2.7 REST 16

2.7.1 HTTP-Methoden 17

2.7.2 JSON 17

2.7.3 HTTP Status Codes 18

3 Umsetzung **20**

3.1 Planung 20

3.1.1 Scrum 21

3.2 Coding 23

3.2.1 Einstieg in React 23

3.2.2 Prototyping 23

3.2.3 Routing 24

3.2.4 Authentifizierung 26

3.3 Retrospektive und Bugfixing 30

4 Zusammenfassung **31**

4.1 Ergebnis & Erfahrungen 31

Literaturverzeichnis **X**

Abbildungsverzeichnis **XII**

Quellcodeverzeichnis **XIII**

1 Einleitung

1.1 Ausgangssituation

Seit Jahren müssen die Mitarbeiter der Firma die Datenbank Einträge sehr umständlich ändern, da kein User Interface dafür vorhanden ist. Unser Ziel war es, genau das zu Ändern. Ich habe gemeinsam mit meinem Team ein Web-Frontend erstellt, welches das Hinzufügen, Bearbeiten und Löschen der Einträge in der Datenbank um ein vielfaches vereinfacht. Nun kann auf das unpraktische Verwenden von Swagger verzichtet werden und auf ein paar wenige Klicks reduziert werden.

Gegeben war bereits ein fertiges Backend mit einer API

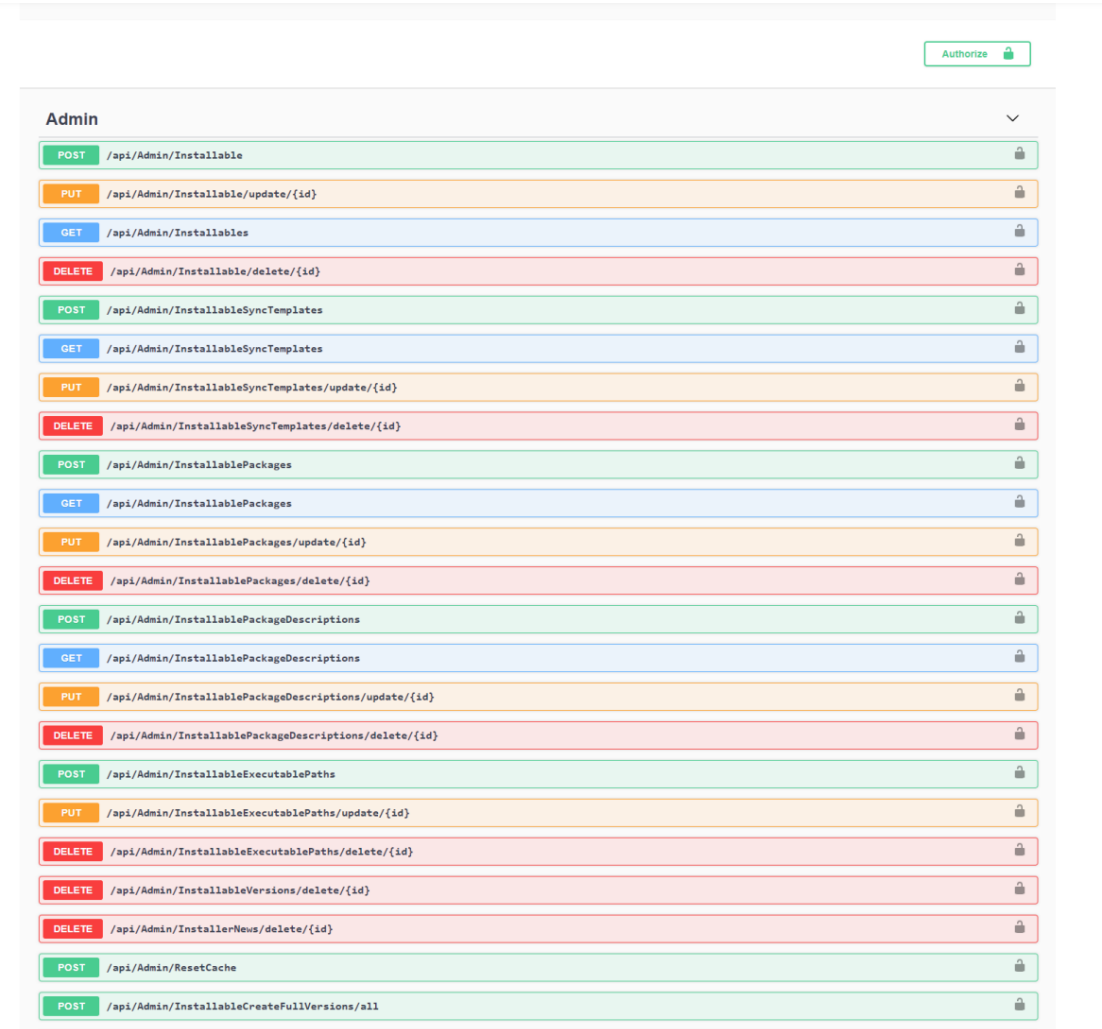

Abbildung 3: Swagger UI - Endpoints

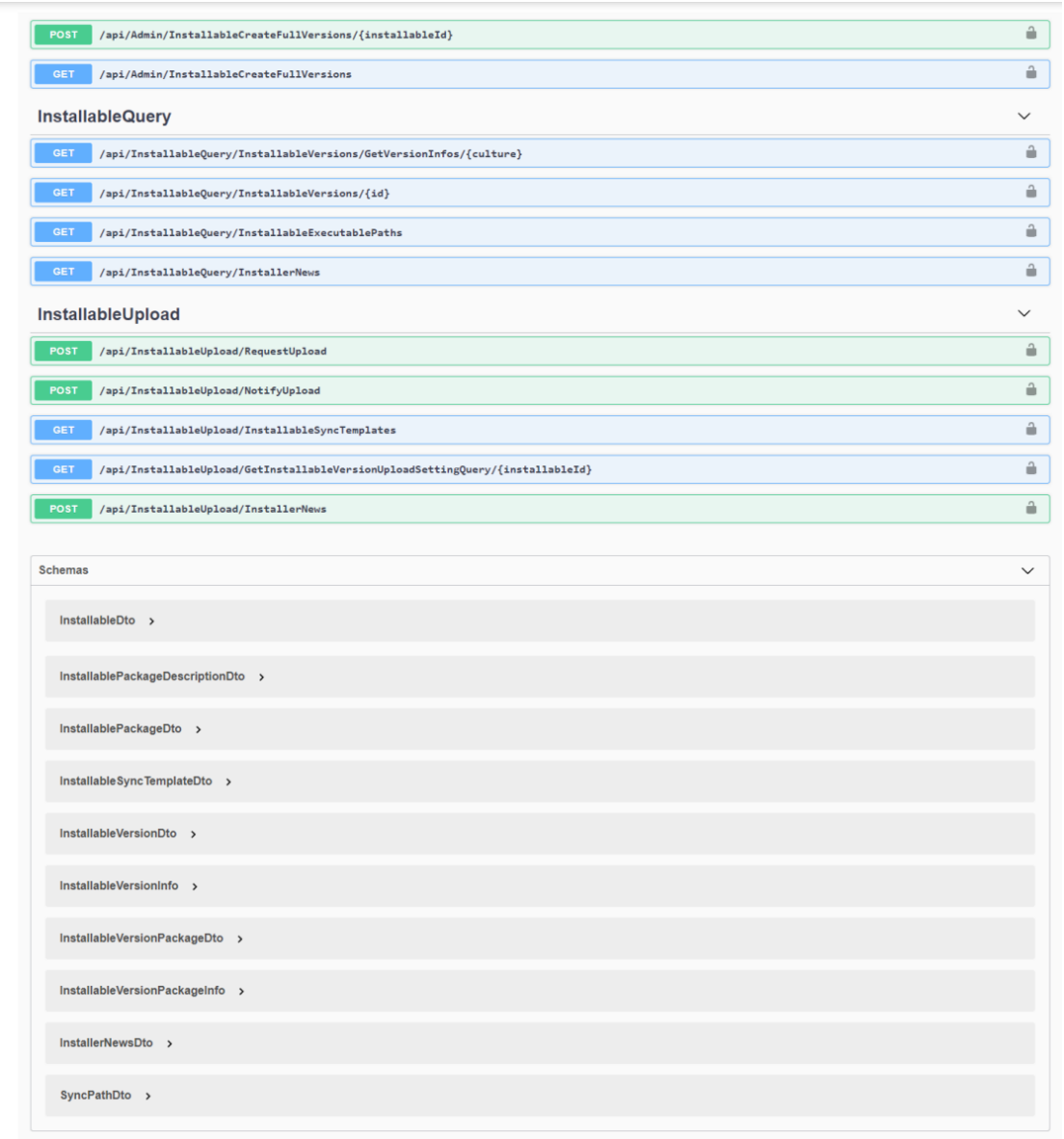

Abbildung 4: Swagger UI - Endpoints

1.2 Aufgabenstellung

Um den Umgang mit der Datenbank für alle Mitarbeiter zu vereinfachen, soll ein leicht verständliches, inuitives Interface entwickelt werden. Die Web-Anwendung darf nur für autorisierte Benutzer in der Abteilung der Firma verfügbar sein. Deswegen ist eine Authentifizierungslogik zu implementieren. Erst nach erfolgreichen Anmelden mit einem gültigen Doka-Account, soll die Website erreichbar sein und Änderungen in der Datenbank durchgeführt werden können.

1.3 Zielsetzung

Die Web-Anwendung soll in einem ansprechenden Design entwickelt werden und ein leicht verständlicher unkomplizierter Umgang muss auch für Nicht-Fachleute gegeben sein.

2 Technologien

2.1 Visual Studio Code 2023

Visual Studio ist eine Entwicklerumgebung von Microsoft. Die aktuellste Version ist Visual Studio 2023. Der einfache aber leistungsstarke Quellcode-Editor, ist für Windows, macOS und Linux verfügbar ist. Er bietet außerdem integrierte Unterstützung für JavaScript, TypeScript und Node.js und verfügt über ein umfangreiches Ökosystem von Erweiterungen für andere Sprachen und Laufzeiten (z. B. C++, C#, Java, Python, PHP, Go, .NET). Visual Studio bietet Syntaxhervorhebung, Code-Vervollständigung, Debugging und viele weitere nützliche Features, die beim Coden unterstützen können. [1]

Bezug auf das Projekt

Für die Entwicklung des Frontends in React wurde Visual Studio Code verwendet. Der Editor ist kostenlos verfügbar, leichtgewichtig und schnell. Er lässt sich schnell öffnen und verfügt über eine gute integrierte Git-Unterstützung. Es lassen sich viele Erweiterungen oder Extensions installieren für die React-Syntax-Hervorhebung, IntelliSense, Debugging und vieles mehr, die das Arbeiten in React enorm verbessern können.

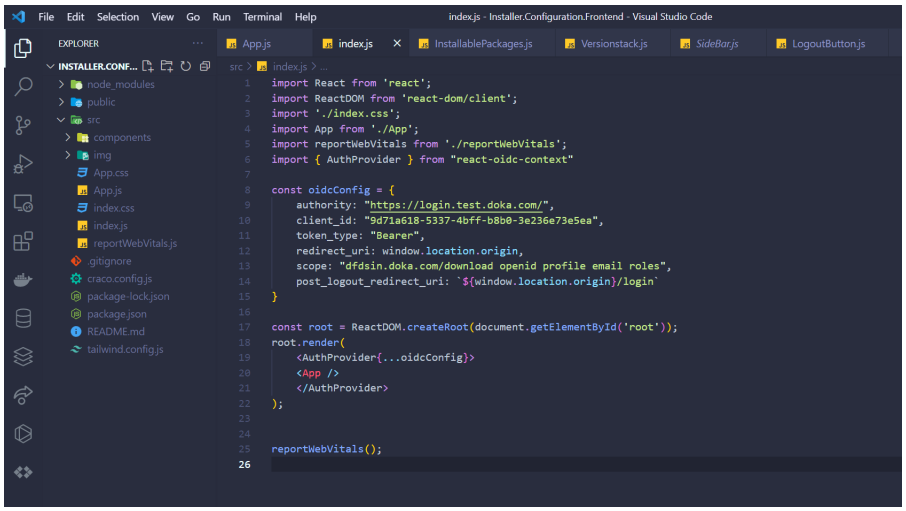

Abbildung 5: Visual Studio Code

2.2 Latex

Normalerweise reichen Textverarbeitungsprogramme wie Word für das Erstellen von Texten aus. Oft aber reicht das nicht für professionelle Texte, wie zum Beispiel für Diplomarbeiten, wo die Ansprüche höher sind. LaTeX erfüllt diese Ansprüche eines Textverarbeitungsprogramms, jedoch zum Preis der Komplexität. Für Neueinsteiger, die davor nur mit Word, LibreOffice und ähnliche gearbeitet haben, ist es anfangs etwas gewöhnungsbedürftig.
Latex kümmert sich um die Typografie, um das Layout und andere Gestaltungsmöglichkeiten, die man zu Beginn festlegen bzw. anpassen kann. Die Idee dahinter ist, dass sich der Autor hauptsächlich auf den Inhalt des Textes konzentrieren kann und sich nicht ständig Gedanken um das Aussehen machen muss. [2]

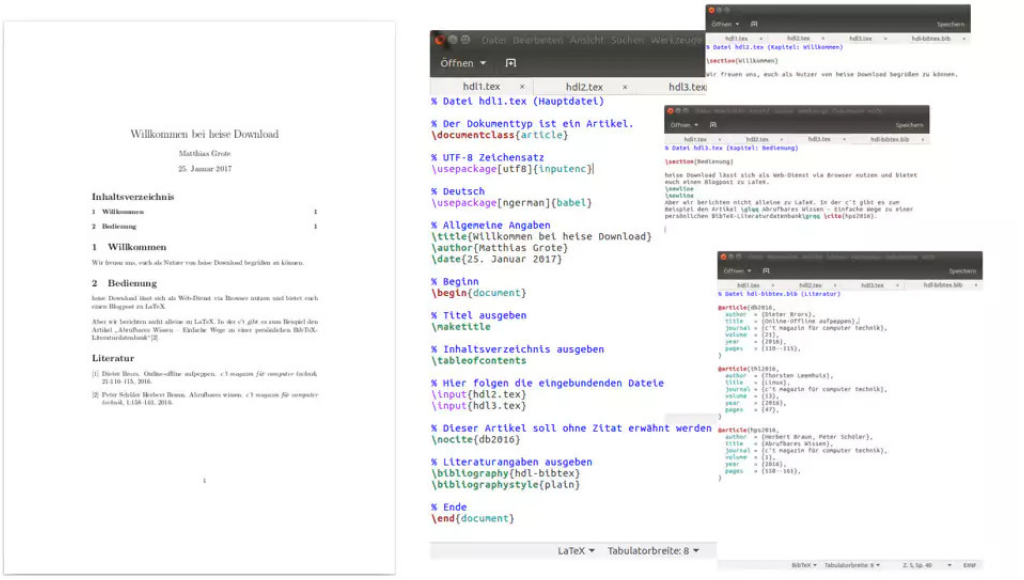

Abbildung 6: Latex [2]

Das Textsatzsystem wurde in Kombinatin mit Visual Studio Code verwendet, um diese Arbeit zu schreiben. Hilfreich dabei war die Visual Studio Erweiterung “LaTex Workshop”, die grundlegende Funktionen und Hilfen inklusive einer Dokumentation, für das arbeiten an LaTex-Dokumenten, bereitstellt.

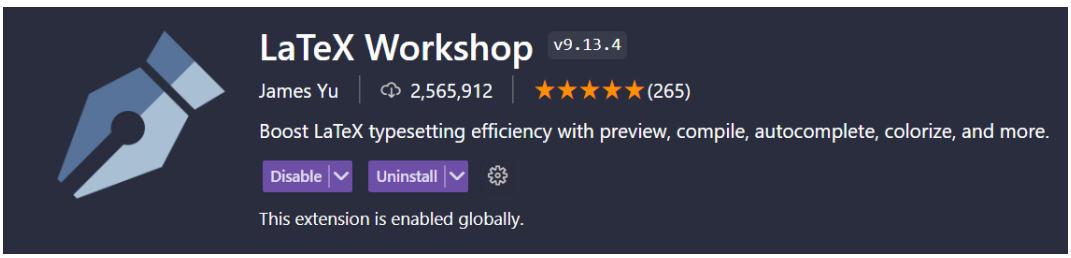

Abbildung 7: Latex-Workshop

Die Erweiterung bietet auch eine Strukturübersicht und hilft beim Entfernen von überflüssigen Files und mehr im “Commands” Abschnitt.

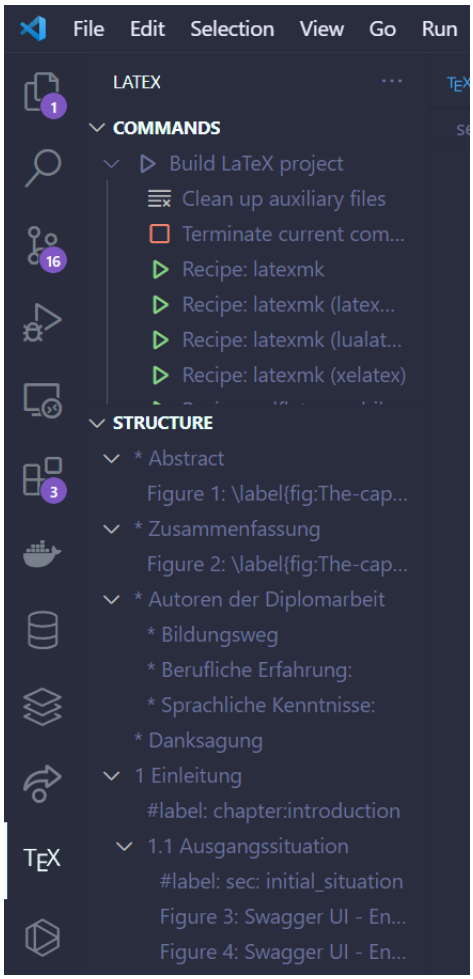

Abbildung 8: Latex-Workshop in VS-Code

2.3 HTML

Die Hypertext Markup Language (HTML) strukturiert textbasierte Inhalte eines Webdokuments. Neben den Inhalten sind auch Meta-Informationen aufzufinden die diese Inhalte beschreiben.

HTML ist einfach zu editieren und kann mit einem beliebigen Texteditor geschrieben werden, da es keine speziellen Anforderungen bei der Erstellung gibt. Oft zu finden im HTML Code sind Tags, die Elemente definieren. Dadurch wird eine Hierarchische Struktur erreicht. Eine HTML-Seite ist meistens zweigeteilt durch den Head-Tag und den Body-Tag.

Listing 1: HTML Beispiel

```
1    <html>
2    <head>
3        <title>My New HTML Page</title>
4    </head>
5    <body>
6      <p>Hello World</p>
7    </body>
8    </html>
```

Head

Dort werden die Meta-Informationen der Seite festgelegt, die nicht im Browser angezeigt werden. Oft werden auch Meta-Tags untergebracht, die für Suchmaschinen relevant sein können.

Body

Hier befinden sich die eigentlichen Inhalte, die durch den Browser visuell dargestellt werden. Beispiele dafür sind, CSS, Mediainhalte, Texthervorhebungen und Überschriften.

2.4 CSS

Während HTML für die Strukturierung und den Inhalt der Website zuständigt ist, sorgen Cascading Stylesheets (CSS) für das Aussehen. Dazu zählen Schriftart, Farbe und Layout. CSS Regeln können mit einem style-Tag im head der HTML Datei festgelegt werden oder in einer externen CSS-Datei. Diese Datei muss mit einem link-Tag in die Seite eingebunden werden.

Listing 2: HTML Beispiel mit CSS

```
1    <head>
2    <link rel="stylesheet" href="style.css">
3
4    <style>
5       body { font-family: Helvetica; }
6    </style>
7  <head>
```

Man kann CSS aber auch mit dem style-Attribut direkt in den HTML-Tags verwenden.

Listing 3: HTML Beispiel mit CSS

```
1    <body style="font-family: Helvetica;">
```

Aufbau

Eine CSS-Regel besteht aus zwei Teilen

- **CSS-Selektor**, also die Bezeichnung für das angesprochene Element (z.B. h1-Tags, div-Tags,...).
- **Eigenschaft und Wert** wie color und green, die dem Element zugewiesen wird.

Die Kombination aus Eigenschaft : Wert steht in geschweiften Klammern und wird durch Semikolons getrennt.

2.5 Tailwind CSS

Tailwind CSS ist CSS-Framework, welches vorgefertigte CSS-Klassen bereitstellt, die verwendet werden können, um HTML-Elemente schnell und einfach zu gestalten. Im Vergleich mit anderen CSS-Frameworks, die vorgefertigte UI-Komponenten bereitstellen, fokussiert sich Tailwind mehr auf Low-Level-Bausteine wie Abstände (Paddings und Margins), Schriftarten und Farbe, um benutzerdefinierte Styles zu erstellen.

Entwickler können vorgefertigte CSS-Klassen hinzufügen, anstatt für jedes Element benutzerdefiniertes CSS zu schreiben 13. Das beschleunigt nicht nur den Entwicklungsprozess, sondern kann sich auch positiv auf die Performance auswirken. Die Dokumentation von Tailwind CSS ist umfangreich und die große Community erleichtert den Einstieg und bietet Unterstützung.[3]

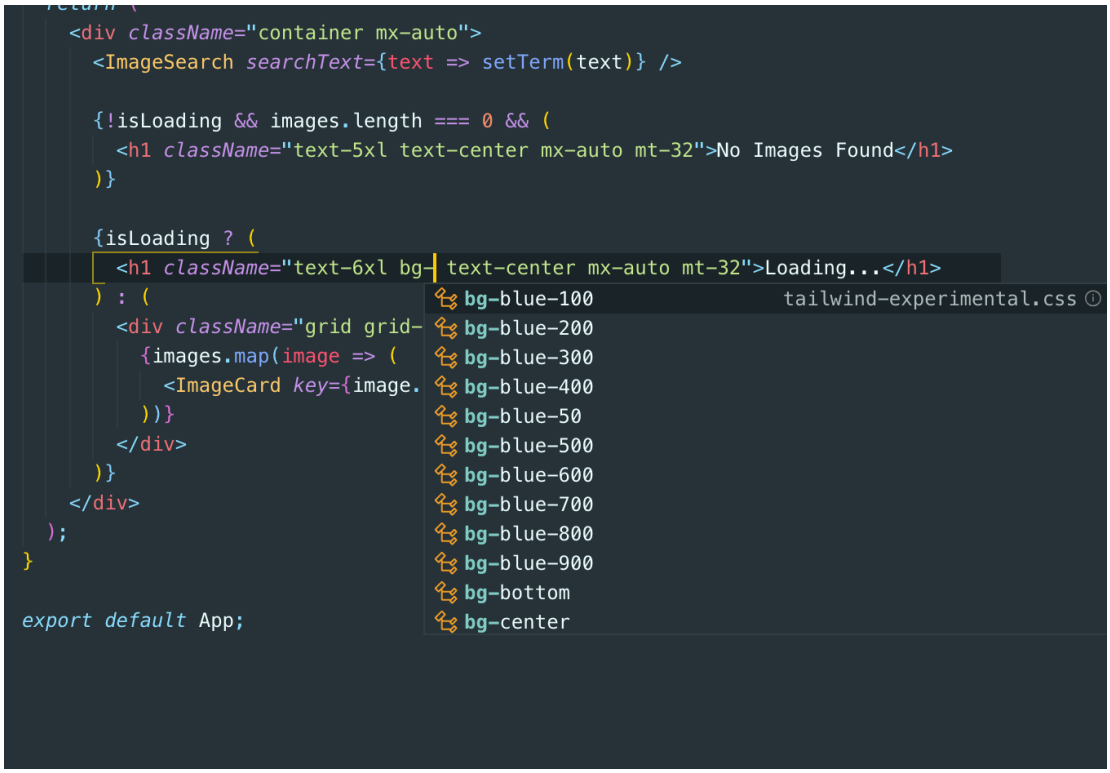

Abbildung 9: Tailwind - vorgefertigte CSS-Klassen [4]

Implementierung

Vorraussetzungen für die Implementierungen sind ein bestehendes Projekt mit einer package.json Datei im Grundverzeichnis. Deswegen sollte natürlich auch Node.js auf dem Rechner installiert sein.

Mit folgendem Befehl wird die neueste stabile Version von Tailwind CSS als Abhängigkeit installiert.

```
npm install -D tailwindcss
```

Abbildung 10: TailwindCSS-Framework installieren

Als nächstes generiert man die Datei tailwind.config.js mit dem Befehl

```
npx tailwindcss init
```

Abbildung 11: Tailwind - config-Datei generieren

Der Inhalt der Datei sieht ohne eigenes Bearbeiten und Hinzufügen so aus.

Abbildung 12: Tailwind - config-Datei

Sobald man die vorgefertigten CSS-Klassen nutzen will oder bereits selbst benutzerdefinierte Klassen selbst hinzugefügt hat, kann man den Erstellungsprozess (build) starten.

```
npx tailwindcss -i ./src/styles.css -o ./public/styles.css --watch
```

Abbildung 13: Tailwind - Erstellungsprozess starten

2.6 React 18.2.0

React ist eine **JavaScript-Programmbibliothek** zur Erstellung von webbasierten Benutzeroberflächen. Jede React-Webanwendung besteht aus wiederverwendbaren Komponenten, die Teile der Benutzeroberfläche bilden - wir können eine separate Komponente für unsere Navigationsleiste haben, eine für die Fußzeile, eine andere für den Hauptinhalt und so weiter.

Diese wiederverwendbaren Komponenten machen die Entwicklung einfacher, weil man wiederkehrenden Code nicht wiederholen muss. Man muss nur die Logik erstellen und die Komponente in jeden Teil des Codes importieren, in dem sie benötigt wird. React ist außerdem eine Single-Page-Anwendung. Anstatt also jedes Mal, wenn eine neue Seite gerendert werden soll, eine Anfrage an den Server zu senden, wird der Inhalt der Seite direkt von den React-Komponenten geladen. Das führt zu einem schnelleren Rendering ohne Neuladen der Seite.

In den meisten Fällen wird für die Erstellung von React-Apps die Syntax JSX (JavaScript XML) verwendet, eine Syntaxerweiterung von JavaScript. Damit kann man die Logik von JavaScript und die Logik der Benutzeroberfläche auf einzigartige Weise kombinieren. Mit JSX muss kein DOM integriert werden, weil man einfach Methoden wie **document.getElementById**, **querySelector** und andere DOM-Manipulationsmethoden verwendet. Die Verwendung von JSX ist zwar nicht obligatorisch, aber sie macht die Entwicklung von React-Anwendungen einfacher.

React kann mit Technologien wie Bootstrap, Tailwind CSS, Firebase und vielen mehr, kombiniert werden. Außerdem kann React auch mit Node.js und anderen Backend-Sprachen verwendet werden, um Full-Stack-Anwendungen und Web-Apps zu erstellen.

Komponenten

In React ist jedes UI-Element eine Komponente. Es gibt zwei Arten um Komponenten: Klassenkomponenten und Funktionskomponenten. Die neue moderne Variante sind Funktionskomponenten, die Hooks als neue Funktion integriert haben. Mehr zu Hooks später. Komponenten haben einen ähnlichen Aufbau wie Funktionen in Javascript und geben einen HTML-Code mit oft dynamischen Werten zurück. Der Name der Komponente muss mit einem Großbuchstaben beginnen, denn sonst würde sie nicht funktionieren.

Das Erstellen einer neuen React-Komponente funktioniert folgendermaßen.

Listing 4: Komponente in React

```
1    function LoginButton {
2      return (
3          <div className="login-container">
4            <button className="login-btn">
5                Authenticate
6            </button>
7          </div>
8      )
9  }
10
11 export default LoginButton
```

In diesem Beispiel wird ein einfacher Button initialisiert. Dieser Button kann dann ganz einfach mit "<LoginButton />" als UI-Element in einer anderen Komponente platziert werden. Wichtig für das Verwenden der Komponente ist das Exportieren in der letzten Zeile im Beispiel "export default LoginButton".

Events

In React werden Events in CamelCase-Syntax geschrieben werden. Das bedeutet, dass zum Beispiel aus "onclick" in einem anderen Framework mit "OnClick" geschrieben wird.

Wenn ein Event als Attribut in einem HTML-Tag übergeben wird, dann verwendet man die geschweiften Klammern: onClickchangeName

Listing 5: React Event

```
1    function LogoutButton() {
2
3      const auth = useAuth();
4
5      const handleClick = () => {
6          auth.removeUser().then(r => console.log(r));
7      }
8
9      return (
10         <button className="logout-btn" onClick={() => handleClick()}>
11             Log out
12         </button>
13     )
14 }
15
16 export default LogoutButton
```

Im obigen Beispiel wird die Funktion "handleClick()" aufgerufen, sobald auf den Button geklickt wird.

Status / State in React

In React ist es so, dass in funktioinalen Komponenten, Änderungen an der Statusvariable nicht im DOM angezeigt werden. Der Status bleibt unverändert.
Deshalb verwendet man die useState Hook. Hooks ermöglichen es auf zusätzliche React-Funktionen zuzugrifen, ohne dafür eine Klasse zu schreiben. Diese Funktionen können beim Verwalten einer Variable behilflich sein.

Listing 6: State von Variablen in React

```
1    const [name, setName] = useState("David");
2    const changeName = () => {
3        setName("Daniel");
4    };
```

Im Beispiel wird eine Statusvariable namens name und eine Funktion - setName - erstellt. Der ursprüngliche Wert der name Variable “David” wird in der useState Hook gespeichert. In der Funktion changeName wird dann die Funktion setName verwendet, um den Wert der name Variable zu ändern.

Hooks

Es gibt natürlich nicht nur die useState Hook in React, sondern sehr viele Andere auf die hier nicht alle eingegangen werden kann. Die useEffect Hook ist eine weitere bekannte und oft verwendete Hook. Der Nutzen kann so erklärt werden, dass jedes Mal ein Effekt ausgeführt wird, wenn sich ein Status ändert. Stanardmäßig wird diese Hook beim ersten Rendering und bei jeder Aktualisierung des Status ausgeführt. Man kann den Effekt jedoch auch an eine entsprechende Statusvariable anhängen.

Listing 7: Hook in React

```
1    function App() {
2      const [day, setDay] = useState(1);
3      useEffect(() => {
4          console.log(`You are in day ${day} of the year`);
5      });
6      return (
7          <div>
8          <button onClick={() => setDay(day + 1)}>Click to increase day</button>
9          </div>
10     );
11 }
12 export default App
```

Beim Klicken des Buttons, wird die Variable day um eins erhöht. Die Änderung des Wertes führt dazu, dass die useEffect Hook ausgelöst wird und die Änderung in der Konsole protokolliert. Das geschieht jedes Mal, wenn sich die Variable ändert.

Props

Um Daten von einer Komponente an eine andere weiterzugeben, werden props verwendet. Das macht den Datenfluss in der App dynamisch. Props ist die Abkürzung für Properties, also Eigenschaften.

Zur Demonstration werden zwei Komponenten benötigt, um Daten von der einen zu anderen Komponente zu übertragen.

Listing 8: Props in React

```
1    function User({name, age}) {
2      return (
3          <div>
4          <p>My name is { name }</p>
5          <p>I am { age } old</p>
6          </div>
7      );
8  }
9  export default User
```

Die User Komponente besitzt in dem Beispiel zwei Parameter (props) deren Wert in einer anderen Komponente bestimmt werden kann.

Listing 9: Props Beispiel 2 in React

```
1    import User from "./User";
2
3    function App() {
4        return (
5            <div className="container">
6              <User name="James" age="19"/>
7            </div>
8        );
9    }
10   export default App;
```

[5]

2.7 REST

REST (Representational State Transfer) wurde entwickelt, um bereits existierende Vorteile bestehender Protokolle zu nutzen. Die Programmierschnittstelle beschreibt einen Ansatz für die Kommunikation zwischen Client und Server in Netzwerken, zum Austausch von Informationen und Daten. Die Daten werden in einer RESTful API in verschiedenen Notationssprachen übermittelt, wobei JSON (JavaScript Object Notation) und XML (Extensible Markup Language) die beiden häufigsten Format sind. Beide Formate sind textbasiert und können vom Menschen leicht gelesen werden.

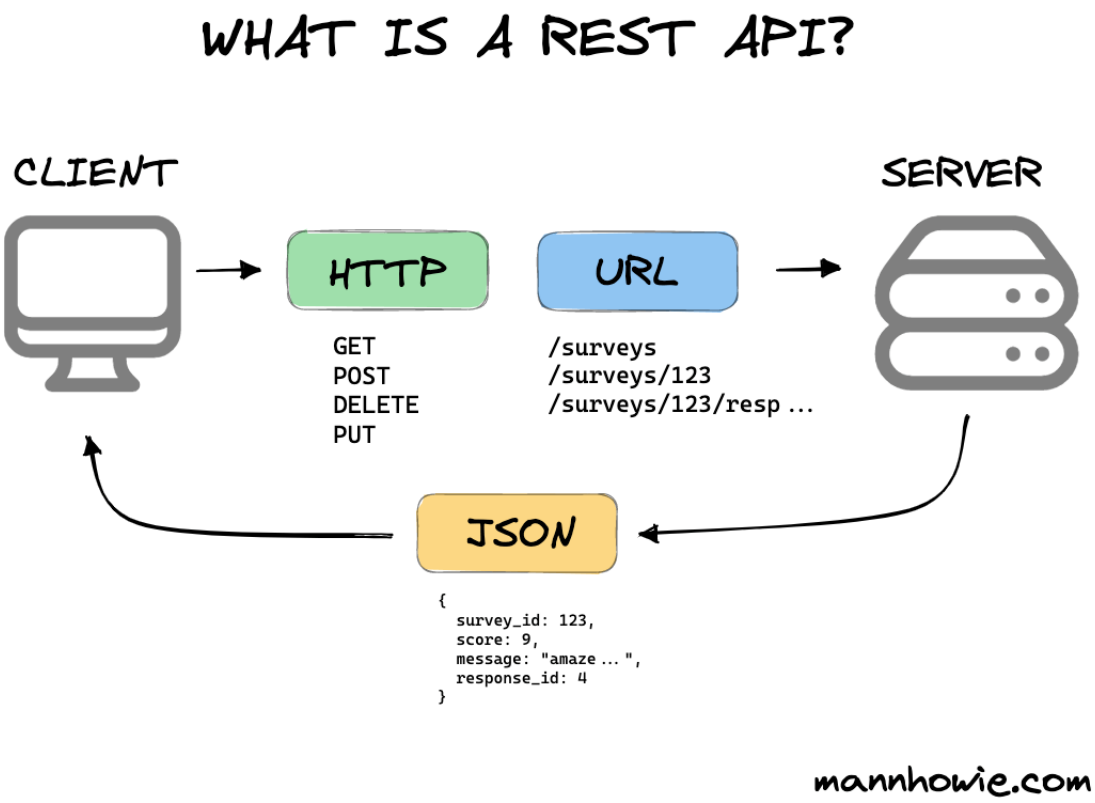

Abbildung 14: REST-API [6]

REST stellt eine Alternative zu anderen Schnittstellen wie SOAP oder WSDL dar, ist dabei aber weder Protokol noch Standard. Die Architektur bedient sich dabei an standardisierte Verfahren wie HTTP, JSON oder XML.
In der Praxis wird Rest bevorzugt mit HTTP(S) realisiert. Die Services werden per URL/URI angesprochen. Die HTTP-Methoden geben an welche Operation von dem Dienst ausgeführt werden soll.[7]

2.7.1 HTTP-Methoden

Das World Wide Web liefert bereits die nötige Infrastruktur für die REST-Schnittstelle, deswegen kann es auch sofort mit einem Browser ausprobiert werden. Zur Hilfe kommen dabei sogennante Fake Online APIs die das Testen der Schnittsetellen vereinfachen. Für das Arbeiten mit RESTful APIs gibt es vier HTTP-Methoden.

- **GET** fordert Daten vom Server an, zum Beispiel alle Benutzer mit dem Namen "John".
- **POST** schickt Daten an den Server, zum Beispiel wenn ein neuer Nutzer angelegt wird.
- **PUT** aktualisiert beziehungsweise ändert bestehende Datensätze auf dem Server. Ein Beispiel dafür wäre, wenn der Name eines Users von "John" auf "Joe" geändert werden soll.
- **DELETE** löscht Daten auf dem Server, zum Beispiel ein inaktiver Benutzer soll entfernt werden.

[8]

2.7.2 JSON

Allgemein

JSON (JavaScript Object Notation) ist ein unabhängiges Datenformat und mit der sehr einfachen menschenlesbaren Struktur, das ideale Format, um Daten zwischen Systemen auszutauschen. Es wird im Unicode Zeichensatz kodiert und wird zwischen Anwendungen immer als Ganzes ausgetauscht. Der Inhalt eines JSON Dokuments ist Objektorientiert aufgebaut und die Datei endet mit ".json".
Die Formatierung eines JSON-Dokuments muss eine genau Struktur haben. Es wird mit { begonnen und mit } beendet. Zwischen den Klammern befinden sich die Inhalte. Die geschweiften Klammern umfassen ein Objekt. Es gibt auch Verschachtelungen in denen im Objekt weitere Objekte definiert werden können. JSON funktioniert mit Key-Value Paaren, das heißt, dass ein Datenfeld immer mit einem Namen eingeleitet wird und nach dem Doppelpunkt folgt ein Wert. [9]

Datentypen

Zeichenketten, sind beliebige Texte der zwischen Anführungszeichen platziert wird. Diese Zeichenkette wird auch String genannt.

Listing 10: JSON Datentyp String

```
1    {
2      "name": "John Doe",
3    }
```

Boolean ist ein Wahrheitswert und kann entweder true oder false annehmen. Der Wert wird ohne Anführungszeichen geschrieben.

Listing 11: JSON Datentyp Boolean

```
1    {
2      "name": "John Doe",
3      "isStudent": true,
4    }
```

Eine Zahl wird mit den Ziffern 0-9 und optional mit einem Punkt und Vorzeichen dargestellt. Auch ein Exponent kann genutzt werden.

Listing 12: JSON Datentyp Number

```
1    {
2      "name": "John Doe",
3      "isStudent": true,
4      "age": 25
5    }
```

Das sind die wichtigsten drei Typen, es gibt natürlich noch viele weitere Möglichkeiten Daten in JSON-Dokumenten darzustellen, wie Objekte, Arrays und mehr. Wenn man ausdrücken will, dass eine Variable leer ist, kann man das mit “null” ausdrücken. [10]

Im Projekt hat die JSON Notation eine Rolle gespielt, weil die Daten die von der API zum Anzeigen ins Frontend geholt wurden im JSON-Format waren, was üblich für den Datenaustausch zwischen API und Anwendung ist.

2.7.3 HTTP Status Codes

Mithilfe von Status Codes, teilt eine REST-API dem Benutzer, das Resultat der Anfrage mit. Die Server Antwort besteht aus einer dreistelligen Zahl und einer kurzen Beschreibung.
Es gibt fünf verschiedene Statusklassen. Die erste Ziffer der steht dabei immer für die jeweilige Klasse.

- **Informative Antworten (100-199)** wie zum Beispiel “102 - In Bearbeitung”.
- **Erfolgreiche Antworten(200-299)** wie “200 - OK”.
- **Umleitungen(300-399)** auf eine andere Adresse. Das Dokument ist auf einer anderen Adresse erreichbar.
- **Client-Fehler(400-499)** , also eine fehlerhafte Anfrage auf Client-Seite wie zum Beispiel “404 - nicht gefunden”.
- **Server-Fehler(500-599)** sind Fehler die auf den Server zurückgehen, wie zum Beispiel “500 - Interner Server-Fehler”.

[11]

3 Umsetzung

3.1 Planung

Sobald die Festlegung und die Aufteilung der Arbeitspakete in der Firma geregelt waren, wurde mit dem Use-Case-Diagramm begonnen. Es haben sich alle Teamkollegen der Diplomarbeit daran beteiligt und nach einigen Änderungen, Diskussionen und Empfehlungen war das Use-Case-Diagramm nach zwei Tagen finalisiert.

Was ist ein Use-Case-Diagramm?

Ein Use-Case-Diagramm ist eine von vielen Arten der Unified Modeling Language (UML). Es wird verwendet um die funktionalen Anforderungen eines Systems beziehungsweise einer Anwendung zu visualisieren. Das Diagramm funktioniert so, dass ein Benutzer mit einem System interagiert, indem es Anwendungsfälle (Use Cases) und Akteure (Benutzer) zeigt, die den Anwendungsfall durchlaufen. Solche Diagramme sind besonders nützlich, für das Verständnis der Funktionalität im Projekt ohne dabei viel technisches Fachwissen vorauszusetzen.[12]

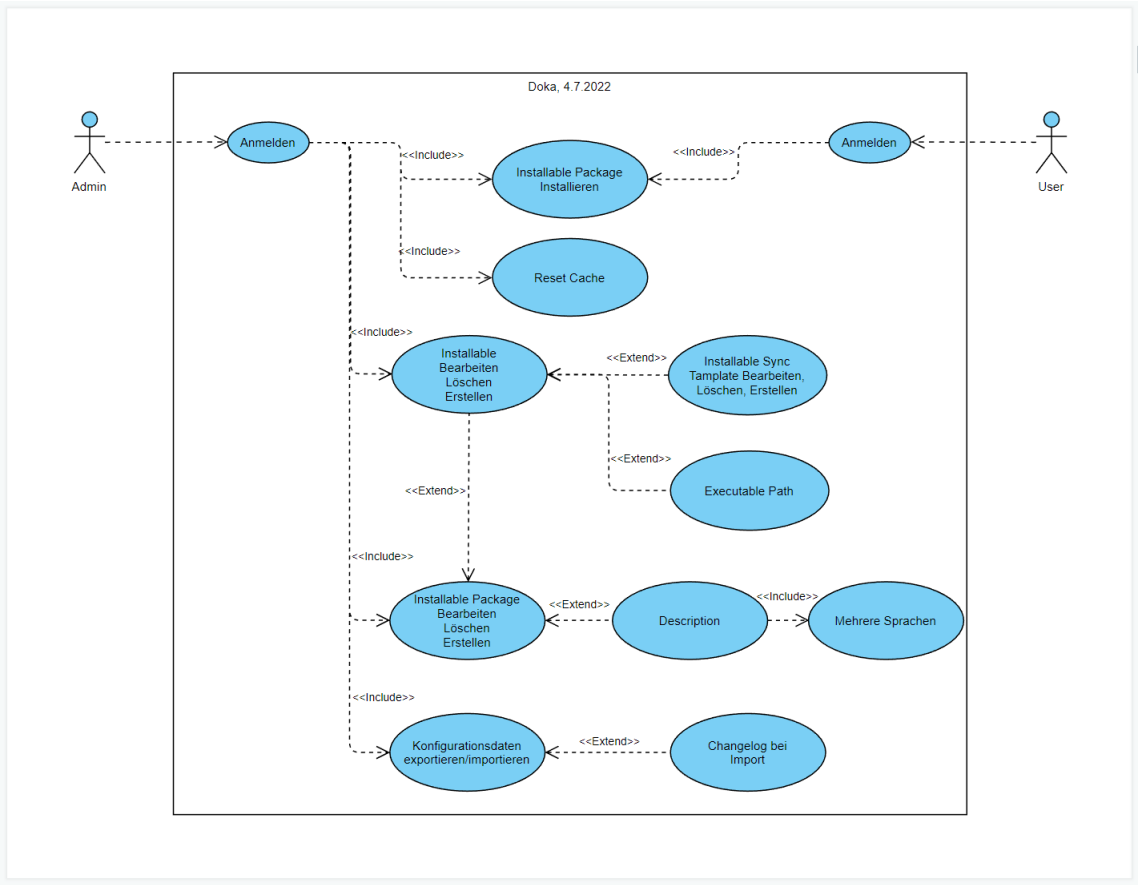

Abbildung 15: Use-Case-Diagramm des Projekts

3.1.1 Scrum

Für den Entwicklungsprozess wurde Scrum als agiles Vorgehensmodell im Projekt angewandt. Scrum ist eine Methode für agiles Projektmanagement. Es wird oft von Softwareentwickler-Teams aber auch von Nichtsoftwareentwicklern für die Zusammenarbeit im Team gewählt. Scrum basiert auf Schlüsselkonzepten, die Struktur und einen klar definierten Arbeitsprozess basierend auf agilen Prinzipien geben.

Scrum Team

Ein Scrm Team besteht aus mehreren Rollen, darunter der Product Owner, der Scrum Master und die Entwickler. Für jeder dieser Rollen ist ein bestimmter Verantwortungsteil im Projekts zugewiesen.

Sprints

Der Zeitraum in dem das Team sich auf die Umsetzung der ausgemachten Aufgaben konzentriert. Ein Sprint dauert normalerweise 2-4 Wochen. Da Scrum ein iterativer, inkrementeller Prozess ist, gibt es während dem Projekt meistens mehrere Sprints mit Arbeitspaketen, die aufeinander aufbauen.

Sprint Planning

Bevor ein neuer Sprint beginnt, wird ein Sprint Planning durchgeführt. Es werden Aufgaben aus dem Product Backlog ausgewählt, die im Zuge des neuenn Sprints zu erledigen sind.

Sprint Review

Die Sitzung in der das Team die Ergebnisse des Sprints präsentiert und ein Feedback erhält.

Product Backlog

Das Backlog ist eine Prioritätsliste von Aufgaben bzw. Funktionen, die eine Anforderung im Projekt darstellen und noch umgesetzt werden müssen.

In diesem Projekt

Mit unserem Projekt, ist das in der Firma genauso abgelaufen. Es wurde zuerst gemeinsam mit den Firmenkollegen, Product Backlog festgelegt und dann nach jedem Sprint in einem Sprint Planning Meeting neue Aufgaben ausgewählt für den nächsten Sprint. [13]

3.2 Coding

3.2.1 Einstieg in React

Die Kollegen in der Firma haben uns das Framework aussuchen lassen, meinten aber dass ihnen das Javascript-Framework React am Liebsten wäre, da es ein Standard in dieser Abteilung des Unternehmens war. Keiner aus unserem Team hatte sich bisher mit React beschäftigt, nichtsdestotrotz haben wir uns für React entschieden, denn es ist immer interessant etwas Neues außerhalb der Schule kennenzulernen und sich neue Dinge selbst beizubringen. Wir haben uns also erstmal mit dem Framework vertraut gemacht, bevor wir mit dem Projekt losstarten konnten.

3.2.2 Prototyping

Der erste Schritt beim Coding, war das Grundgerüst des Designs, begonnen mit einer Navigationsleiste. Das Ganze wurde mit simplem HTML & CSS umgesetzt. siehe 2.3

Tailwind-CSS

Das CSS-Framework Tailwind CSS hat sich als Unterstützung beim Design angeboten, weil es bereits viele vordefinierte CSS-Klassen integriert hat und die Geschwindigkeit und Effizienz stark verbessert hat. Mehr über das Framework in Abschnitt 2.5

3.2.3 Routing

Nach dem Prototyping wurde das Routing für die Website entwickelt.
React benötigt für ein Routing eine externe Library. Diese kann man ganz einfach installieren mit:

```
npm install react-router-dom
```

Abbildung 16: React Router installieren

Als nächstes wird der React Router aufgesetzt. Das ganze findet in dem File App.js statt, was als Basis des Projekts dient.

Listing 13: Routen in React

```
1   function App() {
2     return (
3       <BrowserRouter>
4         <Routes>
5           <Route path="/installables" element={<Installables/>}/>
6           <Route path="*" element={<PageNotFound/>}/>
7         </Routes>
8       </BrowserRouter>
9     );
10  }
```

Im obigen Code wird **Routes** und **Route** von der react-router-dom Library importiert und dann verwendet, um die gewünschte Navigation zu implementieren. Alle **Routen** liegen verschachtelt in den **Routes**. Diese haben zwei Hauptparameter:

- **Path** legt fest, wie der Name schon sagt, auf welchen Pfad der User weitergeleitet werden soll.
- **Element** bestimmt, welche Komponente beim Aufruf des Pfades geladen werden soll. Wenn dieser Parameter nicht ausgefüllt ist, wird ein Fehler auftreten.

Bis jetzt funktioniert das Routing aber nur durch direkter Änderung der URL. Um die verfügbaren Routen auch mit einem Link aufrufen zu können muss **Link** aus der react-router-dom Library importiert werden. Da die Navigationsleiste schon implementiert wurde, kann das Routing per Link gleich in der Navbar-Komponente angelegt werden.

Listing 14: Beispiel für Navigation in React

```
1    const SideBar = () => {
2      return (
3          <div className="nav-container">
4              <Link to="/">
5                  <img className="doka-logo" src={DokaLogo} alt="Doka" />
6              </Link>
7              <Link className="nav-link" to="/installablePackages">
8                  <SideBarIcon />
9              </Link>
10             <Link className="nav-link" to="/import">
11                 <SideBarIcon />
12             </Link>
13         </div>
14     );
15 }
```

Das **to** Attribut legt fest wohin navigiert wird, vorrausgesetzt die Route existiert. siehe Listing 13. Verschachtelt in den **Link-Tags** werden Bilder, Icons oder Komponenten eingebettet um ein visuelles Objekt für den User zum Klicken anzuzeigen.

Man kann den User natürlich auch programmatisch, nach einem Klick auf einen Button zum Beispiel, auf bestimmte Routen weiterleiten mit **navigate()**.

Listing 15: Alternative Navigation

```
1    <button className="btn" onClick={() => navigate('import')}></button>
```

[14]

3.2.4 Authentifizierung

Beim Aufruf der Seite wird als Erstes geschaut ob der User schon angemeldet beziehungswiese authentifiziert ist. Dabei wird geschaut ob der LocalStorage, eine gültige Authentifizierung beinhaltet. Wenn das der Fall ist, dann wird er zur Startseite weitergeleitet.

Wenn der User aber noch nicht authentifiziert ist, dann kann er sich mit einem Login-Button anmelden. Er wird zu einem Login auf **“https://login.test.doka.com/”** weitergleitet. Dort kann er sich mit einem gültigen Doka-Konto anmelden.

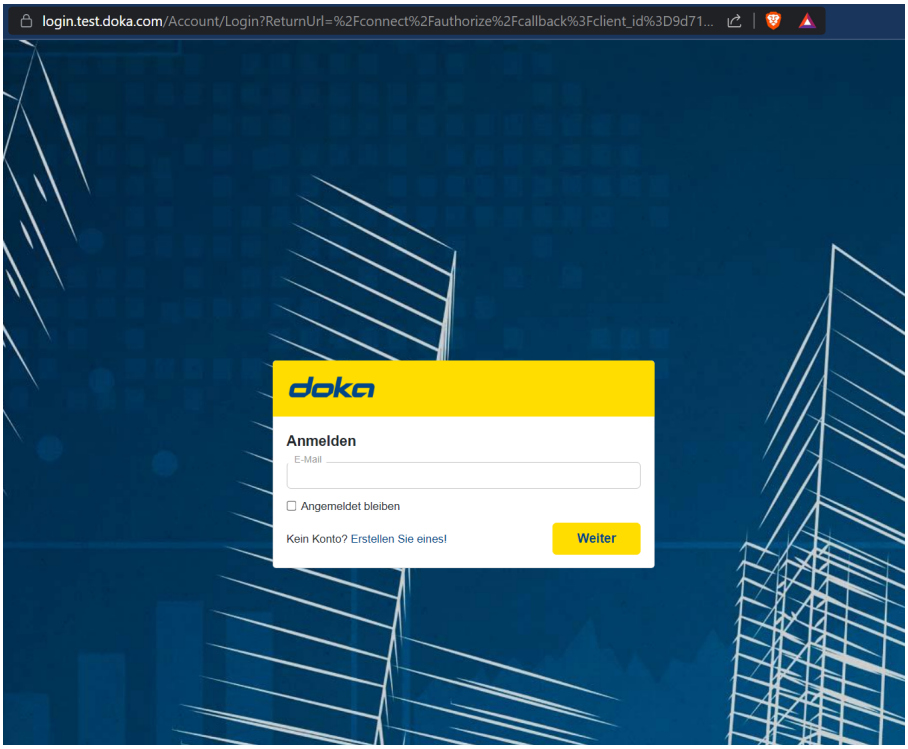

Abbildung 17: Doka Login

Das größte Problem dabei

Das zeitintensivste Problem des ganzen Projektes (zumindest was meinen Teil angeht) war die Authentifizierung hinzubekommen. Grund dafür war die völlig neue Technologie von Bearer Tokens und API-Authentifizierung bishin zu den Libraries mit AuthProvidern in React. Das Grundgerüst für das Ganze war nach eigner Recherche und Besprechung mit den Betreuern relativ schnell implementiert. Anfangs war es jedoch so, dass nach der Anmeldung nicht weitergeleitet wurde zu der React-Page, sondern nur eine leere weiße Seite im Browser zu sehen war. Die Url hat sich natürlich nicht auf "http://localhost:3000/" geändert wie sie sollte, sondern blieb bei der langen Url mit den ganzen Parametern wie client-id, scope, sesssion-state usw.

Die Lösung

Ich habe lange gebraucht um das Probem zu lösen. Es stellte sich heraus, dass ein simpler Timeout von ungefähr 600 millisekunden ausreichen, dass der redirect funktionert. Der Grund dafür ist, dass der Authentifizierungensprozess aufgrund von **asynchroner Verarbeitung** einige Zeit benötigt. Wenn ein direkter Redirect ohne Verzögerung durchgeführt wird, kann es passieren, dass der Prozess noch nicht vollständig abgeschlossen ist. Das Hinzufügen eines Timeouts ermöglicht es also den Authentifizierungsprozess in dieser Zeitspanne abzuschließen, bevor der Redirect ausgelöst wird.

Listing 16: Timeout setzen

```
1    if (window.location.href.includes('code')) {
2              setTimeout(() => {
3                  window.location.href = window.location.origin;
4              }, 600);
5          }
```

Wie funktionert die die Authentifizierung mit Token im Detail?

1. Zuerst muss sich beim Identitätsanbieter authentifiziert werden. Das ganze passiert beim einfachen Anmelden mit einem gültigen Doka-Benutzer. Der Identitätsanbieter stellt dann einen Autorisierungscode aus.

2. Nach dem erfolgreichen Anmelden wird der Benutzer zu der angegeben redirect-uri (In dem Fall "http://localhost:3000") weitergeleitet. Die URL enthält einen Autorisierungscode als Parameter. Dieser wird zusammen mit der client-id dafür verwendet, eine Anfrage an den Identitätsanbieter zu stellen, um einen Zugriffstoken zu erhalten. Dieser Schritt ist jedoch unsichtbar und wird automatisch von der Library erledigt. In dem Fall ist das die Library "react-oidc-context".

3. Nachdem die App den Zugriffstoken erhalten hat, wird speichere ich diesen im Localstorage. Das erlaubt es, auch nach dem Schließen des Browsers zum Beispiel, immer noch angemeldet zu bleiben.

4. Wenn nun eine Anfrage an die geschützte API gesendet wird, fügt man den Bearer Token im Authorization-Header der Anfrage zurück, und der Zugriff ist gesichert.

Listing 17: Authorization-Header

```
1    let httpOptions = {
2      method: 'GET',
3      headers: {
4          Authorization: localStorage.getItem('access_token')
5      }
6  };
```

Wie sieht das ganze im Code aus?

Im index.js File findet die Konfiguration der Authentifizierung statt. Wichtig dabei sind die Login-URL, die client-id, die Art des Tokens der zurückgeliefert werden soll und der redirect, also wohin man weitergeleitet wird bei erfolgreicher Anmeldung. Die App-Komponente mit dem ganzen Inhalt der Website, wird in die AuthProvider-Komponente verschachtelt, in der sich die Authentifizierungslogik befindet.

Listing 18: index.js File mit Auth-Parametern

```
  import { Auth0Provider } from "react-oidc-context"

const oidcConfig = {
    authority: "https://login.test.doka.com/",
    client_id: "9d71a618-5337-4bff-b8b0-3e236e73e5ea",
    token_type: "Bearer",
    redirect_uri: "http://localhost:3000",
    scope: "openid offline_access profile phone email dfdsin.doka.com/download",
    post_logout_redirect_uri: "http://localhost:3000/login"
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <AuthProvider{...oidcConfig}>
    <App />
    </AuthProvider>
);
```

In App.js kann dann der aktuelle Status der Authentifizierungen (isLoading, isAuthenticated,...) abgerufen werden und auf Authentifizierungensmethoden wie **logout** zugegriffen werden. Das ganze funktioniert mit der useAuth-Hook. Das ist dann quasi ein Array aus verschieden Variablen, auf das man über mehrere Komponenten Zugriff hat. Sobald die Anmeldung am Doka-Portal geglückt ist, wird man zur Startseite der Website weitergeleitet und der Bearer-Token wird im LocalStorage gespeichert.

Listing 19: App.js File - Verwalten des Auth-Status

```
  import {useAuth, AuthProvider} from "react-oidc-context"

  function App() {

    const [access_token, setAccess_token] = useState(null);
    const [isAuth, setIsAuth] = useState(false);

    const auth = useAuth();

        if (auth.isLoading) {
            return <div className="box loadingBox"><CircularProgress
                className="loading"/></div>;
        }

        if (auth.isAuthenticated) {
            const token = 'Bearer ' + auth.user.access_token;
            setAccess_token(token);
            localStorage.setItem('access_token', token);

            return (
                <SideBar />
            )
        } else {
            if (window.location.href !== `${window.location.origin}/login`) {
                return window.location.replace(`${window.location.origin}/login`);
            }
        }
```

3.3 Retrospektive und Bugfixing

Da ich bereits 3 Tage vor Ende des Praktikums, mit meinem Teil in React fertig war, bestand meine Aufgabe darin, den Code nach Fehlfunktionen zu überprüfen und noch einige kleine Designänderungen auf Wunsch der Firmenkollegen durchzuführen. Ich habe mich mit ihnen über die Zufriedenheit von dem Projekt und unserer Arbeit während des Praktikums verständigt und eine Retrospektive über alle Sprints des Projekts durchgeführt.

In der übrig gebliebenen Zeit, habe ich meinem Diplomarbeitspartner David Precup mit dem neuen Framework geholfen, der aufgrund eines längeren Krankenstandes, ziemilch weit hinten war mit seinem Teil im Projekt.

4 Zusammenfassung

4.1 Ergebnis & Erfahrungen

Leider können wir das fertige Produkt, aufgrund von internen Firmenkomplikationen nicht herzeigen, sondern nur die Herangehensweisen in der Implementation und Durchführung mit den verwendeten Technologien.

Wir sind mit dem Ergebnis der Diplomarbeit dennoch zufrieden, vorallem wenn man bedenkt, dass wir React als komplett neues Framework gelernt haben, um das Projekt umzusetzen. Das Praktikum im Sommer bei der Doka GmbH, war aufjedenfall eine gute Entscheidung und wir haben viel Erfahrung dadurch mitnehmen können. Nicht nur das Lernen neuer Technologien, sondern auch das Kennenlernen des Arbeitsklimas in einer so großen Firma, haben wir daraus mitgenommen. Obwohl uns natürlich, die Betreuer in der Firma als Hilfe zur Verfügung standen, haben wir einen Großteil durch eigenhändiges recherchieren im Internet bewältigt. Dadurch haben wir bewiesen, dass wir auch ohne Hilfe und außerhalb der Schule, neue technologische Erkenntnisse und Fähigkeiten durch selbstständiges Arbeiten aneignen können.

Literaturverzeichnis

[1] Microsoft, „Visual Studio 2022," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://visualstudio.microsoft.com/de/

[2] heise Download, „Einführung in LaTeX," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://www.heise.de/download/blog/Einfuehrung-in-LaTeX-3599742

[3] Tailwind CSS, „Rapidly build modern websites without ever leaving your HTML," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://tailwindcss.com/

[4] Stack Overflow User Ahmad, 2020, letzter Zugriff am 31.08.2023. Online verfügbar: https://stackoverflow.com/questions/61343447/my-tailwind-css-intellisense-plugin-just-isnt-working-on-my-vscode

[5] Kinsta Wissensdatenbank, „Was ist React.js? Ein Blick auf die beliebte JavaScript-Bibliothek," 2022, letzter Zugriff am 31.08.2023. Online verfügbar: https://kinsta.com/de/wissensdatenbank/was-ist-react-js/

[6] letzter Zugriff am 31.08.2023. Online verfügbar: https://mannhowie.com/rest-api

[7] M.A. Dirk Srocke, „Representational State Transfer (REST), Application Programming Interface (API) - Was ist eine REST API?" 2017, letzter Zugriff am 31.08.2023. Online verfügbar: https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-6e93daacdd9a10179bf0fcd6dec9507d/

[8] „HTTP/Anfragemethoden," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://wiki.selfhtml.org/wiki/HTTP/Anfragemethoden

[9] „Was ist JSON? Praxisnahes Basiswissen," letzter Zugriff am 31.08.2023. Online verfügbar: https://www.opc-router.de/was-ist-json/#:~:text=JSON%20ist%20ein%20komplett%20unabh%C3%A4ngiges,Anwendungen%20immer%20als%20Ganzes%20ausgetauscht

[10] „JSON," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://wiki.selfhtml.org/wiki/JSON

[11] SEO-Küche, „Bedeutung der HTTP Status-Codes," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://www.seo-kueche.de/ratgeber/http-status-codes/#:~:text=Was%20sind%20die%20wichtigsten%20Status,und%20503%20%E2%80%93%20Dienst%20nicht%20verf%C3%BCgbar

[12] „Was ist ein Use Case-Diagramm?" letzter Zugriff am 31.08.2023. Online verfügbar: https://www.microtool.de/wissen-online/was-ist-ein-use-case-diagramm/

[13] Andreas Diehl, „Was ist Scrum?" 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://digitaleneuordnung.de/blog/scrum-methode/#:~:text=Scrum%20ist%20ein%20Rahmenwerk%20f%C3%BCr,basierend%20auf%20agilen%20Prinzipien%20geben

[14] Joel Olawanle, „A Complete Guide to Routing in React," 2023, letzter Zugriff am 31.08.2023. Online verfügbar: https://hygraph.com/blog/routing-in-react

Abbildungsverzeichnis

1 ERD II
2 ERD IV
3 Swagger UI - Endpoints 2
4 Swagger UI - Endpoints 3
5 Visual Studio Code 5
6 Latex [2] 6
7 Latex-Workshop 6
8 Latex-Workshop in VS-Code 7
9 Tailwind - vorgefertigte CSS-Klassen [4] 10
10 TailwindCSS-Framework installieren 11
11 Tailwind - config-Datei generieren 11
12 Tailwind - config-Datei 11
13 Tailwind - Erstellungsprozess starten 11
14 REST-API [6] 16
15 Use-Case-Diagramm des Projekts 21
16 React Router installieren 24
17 Doka Login 26

Quellcodeverzeichnis

1 HTML Beispiel 8
2 HTML Beispiel mit CSS 9
3 HTML Beispiel mit CSS 9
4 Komponente in React 13
5 React Event 13
6 State von Variablen in React 14
7 Hook in React 14
8 Props in React 15
9 Props Beispiel 2 in React 15
10 JSON Datentyp String 18
11 JSON Datentyp Boolean 18
12 JSON Datentyp Number 18
13 Routen in React 24
14 Beispiel für Navigation in React 25
15 Alternative Navigation 25
16 Timeout setzen 27
17 Authorization-Header 28
18 index.js File mit Auth-Parametern 29
19 App.js File - Verwalten des Auth-Status 29