

JavaScript Corona Schnelltest Applikation



Dokumentation

erstellt im Rahmen der Veranstaltung:

Einführung in die Programmierung mit JavaScript

Studiengruppe:	AI_WS20_III
Name Studierender:	Arne Bonn, Sebastian Kirner, Tobias Heise, Cedric Schmitt, Marvin Goinar, Alexander Lehmpfuhl
Anzahl der Wörter: (inkl. wörtliche Zitate / Fußnoten)	3504
Anzahl der Wörter: (exkl. wörtliche Zitate / Fußnoten)	3406
Akademischer Gutachter:	Thomas Strauß
Abgabedatum:	09.03.2022

Inhalt

Use Cases.....	3
Use Case 1: Als Anwender möchte ich mich zum Corona Test anmelden.....	3
Use Case 2: Als Anwender möchte ich eine Startseite mit Informationen zum Testcenter haben.....	5
Use Case 3: Als Verwaltung soll nur ich Zugriff auf die Verwaltungsseite haben.....	9
Use Case 4: Als Verwaltung möchte ich eine Übersicht über alle Anmeldungen haben.....	11
Use Case 5: Als Verwaltung möchte ich eine Suchfunktion für eingetragene Anmeldungen haben	13
Use Case 6: Als Verwaltung möchte ich Neuigkeiten bearbeiten können.....	15
Optionale Funktionen.....	17
Bedienungsanleitung.....	17
Komplettes Projekt starten	17
Nur Frontend starten	17
Nur Backend starten	17
Benutzer/Rollen-Konzept.....	17
API Dokumentation	18
Softwarearchitektur Backend.....	19
Softwarearchitektur Frontend	19
Third Party Software	20
Arbeitszeitaufteilung.....	21
Literaturverzeichnis.....	22

Abbildungsverzeichnis

Abbildung 1: Anmeldemaske.....	3
Abbildung 2: Post Request im Backend.....	4
Abbildung 3: Startseite Corona Testzentrum	6
Abbildung 4: API-Aufruf für Neuigkeiten.....	7
Abbildung 5: Textbox.jsx	7
Abbildung 6:GET-Request: Neuigkeiten	8
Abbildung 7: Login-Seite der Verwaltung.....	9
Abbildung 8: Konfigurationsskript "next-auth".....	10
Abbildung 9: Login-Funktion für OAuth Anbieter	10
Abbildung 10: Weiterleitungs-Funktion für nicht angemeldete Benutzer.....	10
Abbildung 11: Anmeldungsübersicht Frontend	11
Abbildung 12: Asynchrones Laden der Anmeldung	11
Abbildung 13: Dynamisches Erstellen der Anmeldungstabelle.....	12
Abbildung 14: GET-Request für Endpunkt "Anmeldung" im Backend	12
Abbildung 15: Gefilterte Abbildung.....	13
Abbildung 16: Code für Suchfunktion	13
Abbildung 17: Beispiel Neuigkeit Datenstruktur	16
Abbildung 18: Vollständiges Architekturdiagramm Frontend.....	20

Use Cases

Use Case 1: Als Anwender möchte ich mich zum Corona Test anmelden.

Beschreibung

Der Anwender soll im Frontend die Möglichkeit haben, sich für einen Coronatest anzumelden. Hierbei sollen die Stammdaten in einzelnen Eingabefelder eingegeben und validiert werden.

Beteiligte

An dem Use Case ist die Testperson als Anwender, das Frontend als Eingabemaske und das Backend, welches die eingegebenen Stammdaten des Anwenders vom Frontend bekommt, beteiligt.

Scope

Implementiert wird eine Eingabemaske, mit der der Anwender seine Stammdaten an das Testzentrum übermitteln kann. Hierbei sind alle Eingabefelder Pflichtfelder. Es wird zusätzlich für jedes Eingabefeld eine individuelle Prüfung der eingegebenen Daten vorgenommen. Eine Anmeldung kann jedoch nur für den momentanen Tag vorgenommen werden. Außerdem soll eine Stornierung nach einer Anmeldung seitens der zu testenden Person nicht möglich sein.

Frontend

Die Abbildung 1 zeigt die Benutzeroberfläche, in der der Anwender seine Stammdaten eingibt.

Die einzelnen Eingabefelder werden in der „Anmeldung.jsx“ definiert. In die Eingabefelder kann der Anwender durch Linksklick seine Eingaben tätigen und mit einem Linksklick auf den „Buchen“-Button an das Testzentrum schicken. Hat der Anwender beim Ausfüllen seiner Stammdaten Felder vergessen oder fehlerhafte Eingaben getätigt, so werden die Felder rot markiert und unter dem Feld wird in roter Schrift auf den Fehler des Anwenders hingewiesen. Die Überprüfung auf fehlerhafte Daten erfolgt durch individuelle Regulare Ausdrücke. Durch eine Änderung der Eingabe verschwindet der Hinweis auf eine fehlerhafte Eingabe. Hat der Anwender alle Daten erfolgreich eingeben und klickt auf „Buchen!“, wird der Anwender auf die Startseite weitergeleitet und es öffnet sich ein Pop-up mit dem Hinweis, dass die Anmeldung erfolgreich übermittelt wurde. Die Übermittlung findet mittels eines POST-Requests statt. Möchte der Anwender seine Eingabe abbrechen, so kann er mithilfe eines Buttons links auf die Startseite zurückkehren. Seine getätigten Eingaben werden bei diesem Vorrang nicht gespeichert.

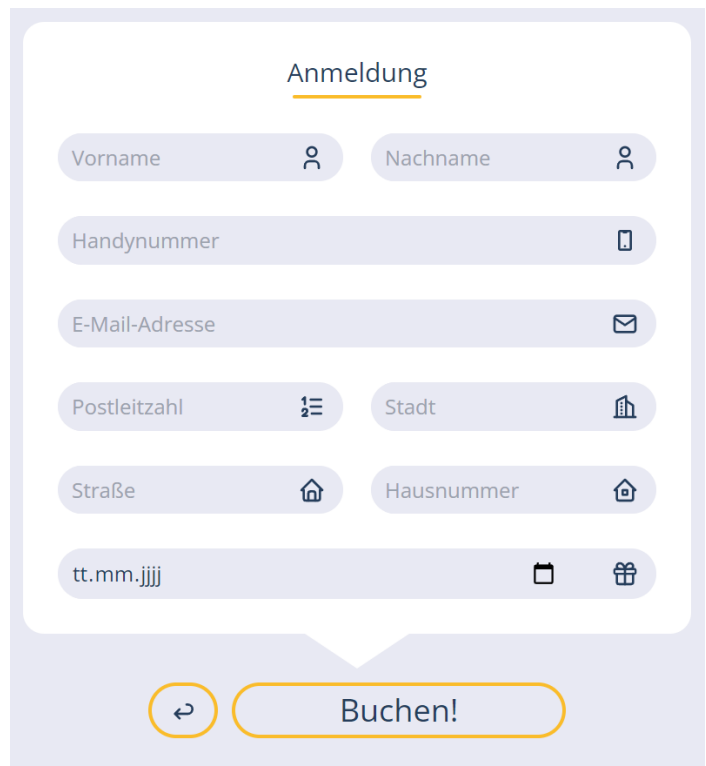


Abbildung 1: Anmeldemaske

API

Damit die Eingabedaten übermittelt werden, wurde ein Endpunkt im Backend geschrieben. Bei dem Endpunkt handelt es sich um einen POST-Request, der über „/anmeldung“ aufgerufen wird. Innerhalb des POST-Requests wird durch die „checkIfFieldsFilled“-Methode überprüft, ob alle Eingabefelder vom Anwender befüllt wurden und ob gültige Symbole verwendet wurden. Ist dies der Fall, werden dem Array die Attribute „dateRegistered“, bestehend aus dem Zeitpunkt der Anmeldung und einer „testId“, bestehend aus einem von der Library „uniquid“ zufällig erstellten String angehängt und in der „anmeldungen.json“ gespeichert. Ist eine Eingabe ungültig, bekommt der Anwender einen HTTP-Statuscode 400 im Falle eines ungültigen Objektes oder einen Statuscode 500 im Falle eines Serverfehlers zurück.

```
app.post("/anmeldung", (req, res) => {
  try {
    if (checkIfFieldsFilled(req.body)) {
      anmeldungen.push(
        "/anmeldungen[]",
        {
          ...req.body,
          dateRegistered: new Date(),
          testId: uniquid("test_"),
        },
        true
      );
    } else {
      throw new Error("Invalid Object");
    }
    res.sendStatus(201);
  } catch (err) {
    console.error(err);
    if (err.message === "Invalid Object") {
      res.sendStatus(400);
    } else {
      res.sendStatus(500);
    }
  }
});
```

Abbildung 2: Post Request im Backend

Use Case 2: Als Anwender möchte ich eine Startseite mit Informationen zum Testcenter haben

Beschreibung

In der Ausgangssituation befindet sich der Anwender, welcher die URL des Corona Testzentrums aufruft. Mit dem Aufruf der Startseite erhält der Anwender im Body der Webseite verschiedene Textboxen mit Informationen über das Testzentrum. Diese Informationen können unter anderem Öffnungszeiten, allgemeine Informationen oder Neuigkeiten sein, die dem Nutzer eine Übersicht zum Testzentrum geben. Im Nachfolgenden werden die empfangenen Informationen als Neuigkeiten bezeichnet.

Zusätzlich zu den Informationen auf der Startseite findet der Anwender im Header der Startseite einen Button. Dieser Button leitet den Anwender weiter auf die Unterseite, auf der er sich anmelden kann (siehe Use Case 1).

Am Ende der Startseite befindet sich ein „Footer“, welcher Informationen über das Team beinhaltet. Zudem befindet sich dort der Link, über den sich Mitarbeiter der Verwaltung anmelden können (siehe Use Case 3).

Beteiligte

Zu den Beteiligten gehört die zu testende Person. Diese Person möchte eine Übersicht der aktuellen Informationen über das Testzentrum und eine Weiterleitung zur Anmeldung. Das Backend der Anwendung ist insofern involviert, da über einen API-Endpunkt die Informationen geladen werden.

Scope

Die Funktion dieses Use Cases ist die einheitliche Darstellung der Informationen über das Testzentrum. Die einzelnen Textboxen auf der Startseite können vom Anwender nicht angeklickt werden und stellen dementsprechend keine Unterseiten für weitere Informationen dar. Zudem werden diese Informationen nicht in Echtzeit aktualisiert. Hierfür muss der Nutzer die Seite neu laden.

Frontend

Abbildung 3 zeigt die Startseite des Corona Testzentrums. Hier werden Informationen wie Öffnungszeiten und News über das Testzentrum dargestellt. Zudem befindet sich im Header der Startseite ein Button, welcher den Nutzer zur Anmeldeseite weiterleitet.

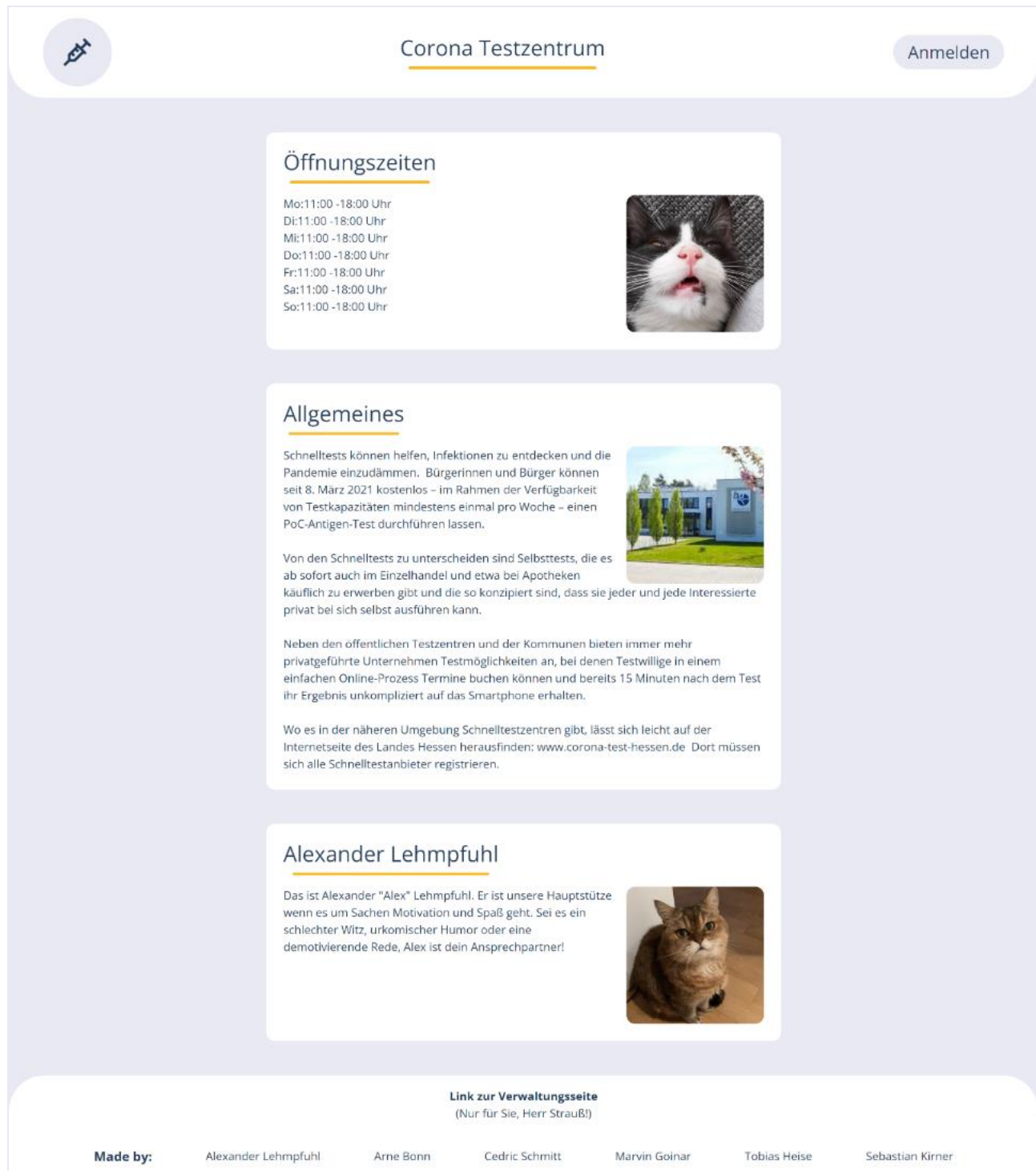


Abbildung 3: Startseite Corona Testzentrum

Die Informationen über das Testzentrum werden in der Mitte der Startseite in einheitlichen Textboxen dargestellt. Optional können Bilder hinzugefügt werden. Der Inhalt der Informationen wird über die API vom

Backend zur Verfügung gestellt und mithilfe der Komponente „Textbox“ einheitlich dargestellt (Abbildung 5). In Abbildung 4 ist der API-Aufruf zu erkennen, über den die Neuigkeiten geliefert werden. Für jeden Eintrag wird anschließend eine Textbox mit dem jeweiligen Inhalt erstellt.

```
export default function Home() {
  const [textboxes, setTextboxes] = useState([]);
  const { setNotificationBar } = useContext(OverlayContext);
  useEffect(() => {
    requestHandler
      .getNeuigkeiten()
      .then(neuigkeiten => {
        setTextboxes(neuigkeiten);
      })
      .catch(err => {
        console.error(err);
        setNotificationBar({ content: `Fehler! Fehlernachricht: "${err}"`, error: true });
      });
  }, []);

  return (
    <>
      <Head>
        <title>Startseite</title>
        <meta name="description" content="Generated by create next app" />
      </Head>

      <main className="min-h-screen">
        <Header>Corona Testzentrum</Header>
        <div className="w-1/2 mx-auto flex flex-col">
          {textboxes.map((el, i) => (
            <Textbox key={`_${el.title}_${i}`} title={el.title} content={el.content} picture={el.picture} />
          ))}
        </div>
      </main>
    </>
  );
}
```

Abbildung 4: API-Aufruf für Neuigkeiten

```
import CustomTitle from "../CustomTitle";
import requestHandler from "../functions/RequestHandler";

const Textbox = (props) => {
  return (
    <div className="bg-white p-6 mt-12 last:mb-12 rounded-xl inline-block">
      <CustomTitle>{props.title}</CustomTitle>
      <div className="mt-4">
        {!!props.picture && (
          <img className="h-48 w-48 rounded-xl object-cover float-right ml-4" src={` ${requestHandler.url}/${props.picture}`} alt={props.picture} />
        )}
        <span className="whitespace-pre-wrap">{props.content}</span>
      </div>
    </div>
  );
};

export default Textbox;
```

Abbildung 5: Textbox.jsx

API

Damit die Informationen über das Testzentrum auf der Startseite angezeigt werden können, müssen sie vorerst vom Backend geladen werden. Dafür gibt es den GET-Request „/neuigkeiten“. Über diesen Endpunkt wird eine JSON Datei gesendet, welche alle Informationen über das Testzentrum beinhaltet. Falls diese Datei leer ist oder nicht vorhanden ist, wird ein Error ausgegeben und der Status Code 500 gesendet. Folgender Code Ausschnitt zeigt den beschriebenen Endpunkt im Backend.


```
app.get("/neuigkeiten", (req, res) => {  
  try {  
    if (neuigkeiten.exists("/neuigkeiten")) {  
      res.json(neuigkeiten.getData("/neuigkeiten"));  
    } else {  
      res.json([]);  
    }  
  } catch (err) {  
    console.error(err);  
    res.sendStatus(500);  
  }  
});
```

Abbildung 6: GET-Request: Neuigkeiten

Use Case 3: Als Verwaltung soll nur ich Zugriff auf die Verwaltungsseite haben

Beschreibung

Die Ausgangssituation ist die Verwaltung, die den Verwaltungsbereich aufruft. Durch einen Login soll verhindert werden, dass Außenstehende, die sich im Netz des Testzentrums befinden, Zugriff auf die Verwaltungsoberfläche haben. Dadurch sind die persönlichen Daten der Patienten sicher und es wird verhindert, dass nicht autorisierte Personen Änderungen an den Testdaten vornehmen können.

Es wurde OAuth integriert, um möglicherweise ein bereits vorhandenes, zentrales Authentifizierungssystem, wie zum Beispiel einen Keycloak Server, zu verwenden.

Wird die Verwaltungsseite unangemeldet aufgerufen, soll man auf die Anmeldungsseite weitergeleitet werden, ohne dass Daten angezeigt werden. Ist man allerdings angemeldet, soll auf die Übersichtsseite der Verwaltung weitergeleitet werden.

Beteiligte

An diesem Use Case ist die Verwaltung, das Frontend und mögliche (externe) Authentifizierungsanbieter beteiligt.

Scope

Das Anmelden mittels Benutzernamen und Passworts ist in unserer Applikation nicht vorgesehen. Die einzige Anmeldemöglichkeit ist zunächst nur mit OAuth via GitHub.

Ebenso ist aktuell keine Multifaktorauthentifizierung gegeben. Falls der Login via Benutzername und Passwort aber integriert wird, sollte ebenfalls eine Multifaktorauthentifizierung implementiert werden.

Frontend

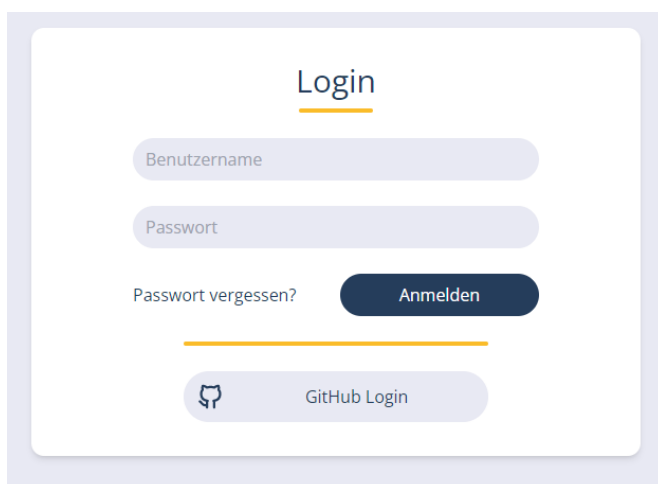


Abbildung 7: Login-Seite der Verwaltung

Um eine Anmeldung über einen OAuth-Anbieter zu ermöglichen, muss zuerst dieser Anbieter in dem Package „next-auth“ eingerichtet werden:

```
import NextAuth from "next-auth";
import GithubProvider from "next-auth/providers/github";

export default NextAuth({
  // Configure one or more authentication providers
  providers: [
    GithubProvider({
      clientId: process.env.GITHUB_ID,
      clientSecret: process.env.GITHUB_SECRET,
    }),
    // ...add more providers here
  ],
});
```

Abbildung 8: Konfigurationsskript "next-auth"

Danach muss eine von „next-auth“ importierte „signIn()“ Funktion aufgerufen werden:

```
<button
  className="bg-ghostwhite flex items-center gap-3 w-1/2 rounded-full px-3 py-2 outline-none relative"
  onClick={() =>
    signIn("github", {
      callbackUrl: `${window.location.origin}/verwaltung`,
    })
  }
  role="link">
  <BrandGithub size={32} color={"#253D5B"} /> <span className="w-full text-center">GitHub Login</span>
</button>
```

Abbildung 9: Login-Funktion für OAuth Anbieter

Falls keine Session gegeben ist, man also nicht eingeloggt ist, wird eine besondere Funktion von „Next.js“ zum Weiterleiten verwendet:

```
export async function getServerSideProps(context) {
  const session = await getSession(context);

  if (!session) {
    return {
      redirect: {
        destination: "/verwaltung/login",
        permanent: false,
      },
    };
  }

  return {
    props: { session },
  };
}
```

Abbildung 10: Weiterleitungs-Funktion für nicht angemeldete Benutzer

Use Case 4: Als Verwaltung möchte ich eine Übersicht über alle Anmeldungen haben

Beschreibung

Die Verwaltung soll eine Seite haben, auf der alle Anmeldungen für den aktuellen Tag eingesehen werden können. Für jede Anmeldung soll sofort der Name, das Geburtsdatum, die Handynummer und die E-Mail-Adresse der Testperson sichtbar sein.

Beteiligte

An diesem Use Case ist die Verwaltung, das Frontend der Verwaltung sowie das Backend beteiligt.

Scope

In diesem Use Case wird das Anzeigen der Anmeldungen behandelt. Das Filtern der Anmeldungen nach einem Suchbegriff wird in Use Case 5 umgesetzt. Das Löschen und Bearbeiten von Anmeldungen, das Anzeigen aller vorhandenen Daten über eine Anmeldung und das Wechseln zwischen verschiedenen Tagen werden nicht umgesetzt. Für diese Funktionen wird ein Button ohne Funktionalität in der Benutzeroberfläche hinzugefügt.

Frontend

Folgende Abbildung zeigt die Benutzeroberfläche, mit der die Verwaltung alle vorhandenen Anmeldungen einsehen kann. Die Funktionalität ist, wie im Scope beschrieben, nur teilweise umgesetzt.



Abbildung 11: Anmeldungsübersicht Frontend

Die benötigten Daten für die Anmeldungen werden asynchron vom Backend angefordert.

```
const [registrations, setRegistrations] = useState([]);
const [searchString, setSearchString] = useState("");

useEffect(() => {
  requestHandler.getAnmeldungen().then(registrations => {
    setRegistrations(registrations);
  });
}, []);
```

Abbildung 12: Asynchrones Laden der Anmeldung

Sobald die Daten vom Backend geladen wurden, werden diese in die Tabelle gemappt.

```
<table className="table-fixes w-full mt-6">
  <thead className="text-left">
    <tr>
      <th>Name</th>
      <th>Geburtsdatum</th>
      <th>Handynummer</th>
      <th>E-Mail</th>
      <th className="w-24"></th>
    </tr>
  </thead>
  <tbody className="border-mango border-t-4 w mx-auto rounded-full">
    {registrations
      .filter(r => Object.values(r).some(event => event.toLowerCase().includes(searchString.toLowerCase())))
      ?.map(r => (
        <tr key={r.testId}>
          <td className="pt-2">
            {r.surname} {r.lastName}
          </td>
          <td className="pt-2">{new Date(r.birthdate).toLocaleDateString("de-DE")}</td>
          <td className="pt-2">{r.phoneNumber}</td>
          <td className="pt-2">{r.mail}</td>
          <td className="flex gap-1 justify-end pt-2">
            <Trash size={24} strokeWidth={1} color={prussianblue} />
            <Edit size={24} strokeWidth={1} color={prussianblue} />
            <Eye size={24} strokeWidth={1} color={prussianblue} />
          </td>
        </tr>
      )
    )}
  </tbody>
</table>
```

Abbildung 13: Dynamisches Erstellen der Anmeldungstabelle

API

Die Daten der Anmeldungen werden über den Endpunkt „/anmeldungen“ mit einem GET-Request bereitgestellt. Die folgende Abbildung zeigt den Code im Backend. Alle vorhandenen Anmeldungen werden aus der Datenbank geladen und als JSON-Objekte zurückgegeben. Im Falle eines Fehlers wird der HTTP-Statuscode 500 zurückgegeben.

```
app.get("/anmeldung", (req, res) => {
  try {
    if (anmeldungen.exists("/anmeldungen")) {
      res.json(anmeldungen.getData("/anmeldungen"));
    } else {
      res.json([]);
    }
  } catch (err) {
    console.error(err);
    res.sendStatus(500);
  }
});
```

Abbildung 14: GET-Request für Endpunkt "Anmeldung" im Backend

Use Case 5: Als Verwaltung möchte ich eine Suchfunktion für eingetragene Anmeldungen haben

Beschreibung

Die Verwaltung soll im Frontend die Möglichkeit haben, nach Anmeldungen zu suchen. Dazu wird ein Suchfeld über den Anmeldungen angezeigt.

Beteiligte

An diesem Use Case sind die Testpersonen und das Frontend der Verwaltung beteiligt.

Scope

In diesem Use Case wird die Suchfunktion behandelt. Es ist möglich, gezielt nach Anmeldungen zu suchen, da nach jeglichen von den Testpersonen zur Verfügung gestellten Informationen gesucht werden kann.

Frontend

Abbildung 11 zeigt die Benutzeroberfläche der Verwaltung mit allen vorhandenen Anmeldungen.

In das Feld „Suche“ kann der gewünschte Suchbegriff eingegeben werden. Abbildung 15 zeigt das Ergebnis, wenn man nach einem Nachnamen sucht.



Abbildung 15: Gefilterte Abbildung

Ermöglicht wird die Suchfunktion durch den Code in Abbildung 16:

```
{registrations
  .filter(r => Object.values(r).some(event => event.toLowerCase().includes(searchString.toLowerCase())))
  ?.map(r => (
    <tr key={r.testId}>
      <td className="pt-2">
        {r.surname} {r.lastName}
      </td>
      <td className="pt-2">{new Date(r.birthdate).toLocaleDateString("de-DE")}</td>
      <td className="pt-2">{r.phoneNumber}</td>
      <td className="pt-2">{r.mail}</td>
      <td className="flex gap-1 justify-end pt-2">
        <Trash size={24} strokeWidth={1} color={prussianblue} />
        <Edit size={24} strokeWidth={1} color={prussianblue} />
        <Eye size={24} strokeWidth={1} color={prussianblue} />
      </td>
    </tr>
  ))
})}
```

Abbildung 16: Code für Suchfunktion

Die Funktion verwandelt die Anmeldedaten in ein Array, welches nach dem eingegebenen Begriff durchsucht wird. Sollte der Begriff nicht im Array vorhanden sein, wird er entfernt und nicht mehr angezeigt. Um Groß- und Kleinschreibung zu umgehen, werden sowohl das Array, als auch der Suchbegriff in Kleinbuchstaben umgewandelt.

Use Case 6: Als Verwaltung möchte ich Neuigkeiten bearbeiten können

Beschreibung

Die Ausgangssituation ist, dass ein Mitglied der Verwaltung sich bereits eingeloggt hat und die Neuigkeiten bearbeiten möchte. Das Ziel dieses Use Cases ist es, ein möglichst einfaches und übersichtliches User Interface zur Verfügung zu stellen.

In diesem Use Case kann die Verwaltung Neuigkeiten, die auf der Startseite der Kunden angezeigt werden, bearbeiten. Hierbei wird nicht direkt von dem Frontend der Verwaltung auf das Frontend der Kunden zugegriffen, sondern es werden die Daten, die auf dem Backend liegen, bearbeitet.

Beteiligte

In dem Use Case ist das Frontend und das Backend beteiligt. Im Frontend kann der Anwender die Neuigkeit bearbeiten und das Backend empfängt die Daten.

Scope

Wenn die Ausgangssituation erreicht ist, soll das Verwaltungspersonal in einer Leiste einen Reiter aufrufen können, über den es eine Liste mit den aktuellen Neuigkeiten angezeigt bekommt. Diese kann er dann mittels Symbolen rechts in der jeweiligen Reihe bearbeiten oder löschen. Außerdem ist in jeder Reihe sowohl der Titel der Neuigkeit, als auch eine Vorschau des Textes vorhanden. Zusätzlich kann durch einen Button über der Tabelle ein neuer Eintrag hinzugefügt werden. Wenn der Button gedrückt wird, öffnet sich ein Modal, das leer ist. Klickt man auf das „Bearbeiten“ Symbol, ist das sich öffnende Modal mit dem alten Text vorausgefüllt. Hier gibt es die Eingabefelder „Inhalt“, „Titel“ und „Bild hochladen“. Beim Klick auf den „Bestätigen“ Button wird dann die Neuigkeit mit den eingetragenen Daten aktualisiert, beziehungsweise hinzugefügt.

Es wird jedoch keine Überprüfung, ob alle Felder korrekt befüllt sind, implementiert, sowie keine eventuelle Unterscheidung zwischen verschiedenen Benutzergruppen, die berechtigt wären Neuigkeiten zu veröffentlichen oder nicht.

Frontend Architektur

Beim Klick auf den Neuigkeiten Reiter wird eine neue React Komponente geöffnet, die durch Next.js kontrolliert wird. Somit hat sie ihre eigene Route. Diese Komponente besteht aus einer umrandenden Komponente, die den Header darstellt, mit dem man auch initial den Reiter „Neuigkeiten“ anklicken kann. In der Seite selber wird eine Tabelle, ein Eingabefeld und ein Button erstellt. Der „Neuigkeiten“-Request, mit der die Tabelle initial befüllt wird, wurde mit einer Klasse „RequestHandler“ erstellt. Diese Klasse beinhaltet alle Anfragen, die in dem Projekt vorkommen. Bei Klick auf das Mülleimer-Icon in einer Zeile wird der entsprechende DELETE-Request, der ebenfalls in dem „RequestHandler“ definiert wurde, ausgeführt. Bei Klick auf „Veröffentlichen“ wird ein React Kontext¹ benutzt. Dieser beinhaltet eine Funktion zum Aufrufen eines Modals, bei der eine Komponente als Inhalt des Modals mitgegeben werden kann. Dieses Modal liegt in der Komponentenhierarchie über der eigentlichen Komponente, da hier auch der Kontext initialisiert wurde. Die dem Modal mitgegebene Komponenten werden in dem „Components“ Ordner definiert. Hier werden die Eingabefelder für die Daten der Neuigkeiten definiert. Optional können in die Komponente noch Standardwerte für die Felder mit hineingegeben werden. Das machen wir im Fall des „Bearbeiten“ Symbols, da hier die Werte nur bearbeitet werden müssen. Außerdem wird eine Funktion „onCustomSubmit“ als Property mitgegeben. Diese Funktion wird bei Klick auf den Veröffentlichen Button ausgeführt und beinhaltet im Fall der beiden Aufrufe der Komponenten den entsprechenden API-Call. Außerdem wird der Modal wieder

¹ Literaturverzeichnis [2]

auf seinen geschlossenen Zustand null versetzt, wodurch sich dieses Modal schließt. Auch wird eine zweite Funktion des Kontextes zum Öffnen der „NotificationBar“ mit einer Bestätigung oder einer eventuellen Fehlermeldung ausgeführt.

API

Für diesen Use Case gibt es drei weitere Endpunkte im Backend. Zusätzlich zum GET-Request existiert nun ein POST, ein DELETE und ein PATCH Request, die über den Endpunkt „/neuigkeiten“ aufgerufen werden. Die Anfragen POST und PATCH werden zudem nicht mit einem „application/json“, sondern mit einem „multipart/form-data“ body ausgeführt, sodass einfacher ein Bild mitgegeben werden kann. Eine Besonderheit dieses Formates ist, dass nicht im Header der Anfrage angegeben werden darf, welcher Content-Type verwendet wird, da in dieser Content-Type ein Trenner für die „Form-Data“ durch den Browser gegeben wird. Wie bei allen implementierten Anfragen führt ein Fehler in der Verarbeitung zu einem zurückgegeben HTTP-Statuscode von 500.

POST REQUEST

Mithilfe der POST Request wird den Neuigkeiten ein neuer Eintrag hinzugefügt. Dem „multipart/form-data“ Body werden die benötigten Daten, wie dem Titel, dem Inhalt und dem Dateitypen entnommen. Nun wird der Eintrag mit den Daten erstellt. Außerdem werden in den Eintrag ein zufällig generierter Name für das Bild mit der korrekten Dateiendung und eine zufällig generierte „newsId“ hineingegeben. Diese zufällig generierten IDs werden für die eindeutige Identifizierung des Eintrags, beziehungsweise des Bildes, verwendet. Die Dateistruktur wird mit einem Beispiel in Abbildung 8 dargestellt. Zusätzlich wird das Bild noch unter dem zufällig generierten Namen im Ordner „Global“ für die spätere Weiterverwendung abgespeichert. Außerdem wird die neu generierte „newsId“ als Antwort zurückgegeben, damit sie im Frontend weiterverwendet werden kann.

```
{
  "title": "Allgemeines",
  "content": "Schnelltests können helfen, Infektionen zu entdecken und die Pandemie einzudämmen.",
  "picture": "ba.jpg",
  "newsId": "1233"
},
```

Abbildung 17: Beispiel Neuigkeit Datenstruktur

PATCH REQUEST

Die PATCH Request ist ähnlich zur PUT Request. Jedoch darf das Bild nur dann ersetzt werden, wenn ein neues Bild mitgeliefert wird und in jedem Fall muss der Titel des Bildes neu gesetzt werden, da das Bild entweder neu gesetzt wird oder komplett gelöscht wird bei jeder Anfrage. Außerdem darf keine neue „newsId“ generiert werden. Zudem wird die Neuigkeit ersetzt, die geupdated werden soll. Dafür muss mittels der in der Anfrage vorhandenen „newsId“ die richtige Neuigkeit aus dem vorhandenen Datensatz gefunden werden. Es wird kein Inhalt, sondern nur ein passender Statuscode zurückgegeben.

DELETE REQUEST

In dieser Request wird ein bestimmter Eintrag gelöscht. Die zur Identifikation verwendete „newsId“ wird über die Anfrage-URL mitgegeben. Falls der Eintrag, der gelöscht werden soll, nicht gefunden wird, wird der Status 404 zurückgegeben, andernfalls wird der Status 200 zurückgegeben.

Optionale Funktionen

Ein Testzentrum kann noch viele weitere Funktionen haben, die hier noch nicht implementiert wurden, aber hilfreich für einen aktiven Betrieb sind.

In einem nächsten Schritt könnte man der Verwaltung ermöglichen Testergebnisse in der Verwaltungsübersicht einzutragen. Dadurch wird jedem Datensatz ein Testergebnis zugewiesen, dieses kann „positiv“ oder „negativ“ sein. Mit dieser zusätzlichen Information können viele weitere Funktionen implementiert werden. Zum Beispiel kann aus dem Datensatz automatisiert ein Dokument erstellt werden, welches alle Informationen über die jeweilige Person und dem Testergebnis enthält. Dieses Dokument kann dann nach dem Testen der Person übergeben werden oder in einem weiteren Use Case automatisiert an die hinterlegte E-Mail-Adresse gesendet werden. Des Weiteren können aus den Datensätzen Statistiken erstellt werden, welche für die Verwaltung sichtbar sind. Diese Statistiken können beispielsweise Auskunft darüber geben wie viele Anmeldungen es pro Tag gibt oder wie groß das Verhältnis zwischen positiven und negativen Testergebnissen ist.

Zudem kann der Anmeldevorgang optimiert werden, um die User-Experience zu verbessern. Geplant ist eine API, die nach Eingabe der Postleitzahl die Stadt automatisch ergänzt und im Eingabefeld Straße automatisiert Straßennamen vorschlägt. Dieses Feature verhindert darüber hinaus die Eingabe von nichtexistierenden Adressdaten. Eine weitere Optimierung des Anmeldeprozesses ist eine Bestätigung, ob ein Termin ordnungsgemäß eingetragen wurde. Diese Bestätigung kann ebenfalls per E-Mail erfolgen, nach dem der Anwender seine Anmeldung abgeschickt hat.

Um Personen mit wenig Interneterfahrung nicht auszuschließen, könnte der Verwaltung ermöglicht werden Datensätze vor Ort oder telefonisch für den Kunden einzutragen.

Zusammengefasst sorgen die anfangs beschriebenen Use-Cases für einen reibungslosen Tagesablauf, aber viele Funktionen können noch ergänzt oder optimiert werden, um die User-Experience und die Verwaltungsmöglichkeiten zu verbessern.

Bedienungsanleitung

Für die Installation dieses Projektes muss eine Internetverbindung verfügbar und node installiert sein.

Komplettes Projekt starten

1. Repository herunterladen beziehungsweise Dateien anderweitig entpacken
2. Terminal im Ordner „./frontend“ öffnen
3. „npm run project“ ausführen

Nur Frontend starten

1. Identisch zu Projekt starten
2. Identisch zu Projekt starten
3. „npm run dev“ ausführen

Nur Backend starten

1. Identisch zu Projekt starten
2. Terminal im Ordner „./backend“ öffnen
3. „node ./index.js“ ausführen

Benutzer/Rollen-Konzept

Das Berechtigungskonzept unterscheidet zwischen authentifizierten Benutzern und nicht authentifizierten Benutzern. Ein nicht authentifizierter Benutzer hat Zugriff auf die Startseite, Anmeldeseite und

Verwaltungsloginseite. Auf der Verwaltungsloginseite kann sich der unauthentifizierte Benutzer durch ein „GitHub“-Login zu einem authentifizierten Benutzer „upgraden“. Das Rollenkonzept sieht vor, dass bestimmten GitHub Accounts eine Rolle zugewiesen werden kann, mit der der Account auf die Verwaltungsanmeldungsseite freigeschaltet wird und die Kundendaten einsehen kann. Ein vollständiges Rollenkonzept wurde jedoch noch nicht implementiert.

API Dokumentation

Die API hat zwei Endpunkte, die insgesamt sechs Funktionen bedienen. Es gibt die Endpunkte „/neuigkeiten“ und „/anmeldungen“. Beim Endpunkt „/neuigkeiten“ gibt es die REST-Funktionen GET, POST, PATCH und DELETE.

Die Methoden erfüllen ihre Funktion gemäß dem [REST Standard von Microsoft](#).²

„/neuigkeiten“

Methode	Request	Response	HTTP-Statuscode
GET (application/json)	{}	[{ title: string, content: string, picture: string, newsId: string, }, ...]	Kein Fehler: 200 Serverfehler: 500
POST (multipart/form-data)	(FormData){ title: string, content: string, picture: file, }	{ newsId: string }	Kein Fehler: 201 Serverfehler: 500
PATCH (multipart/form-data)	(FormData){ title: string, content: string, picture: file, newsId: string, }	{}	Kein Fehler: 200 Serverfehler: 500
DELETE (application/json)	{ newsId: string }	{}	Kein Fehler: 200 „newsId“ nicht gefunden: 404 Serverfehler: 500

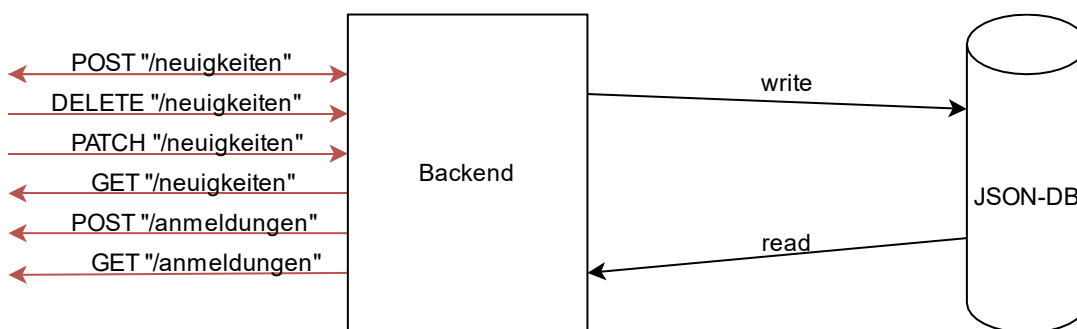
„/anmeldung“

Methode	Request	Response	HTTP-Statuscode
GET (application/json)	{}	[{ surname: string, lastName: string, phoneNumber: string, mail: string, postCode: string, city: string, street: string, }	Kein Fehler: 200 Serverfehler: 500

² Literaturverzeichnis [1]

		houseNumber: string, birthdate: string dateRegistered: string testId: string }, ...]	
POST (application/json)	{ surname: string, lastName: string, phoneNumber: string, mail: string, postCode: string, city: string, street: string, houseNumber: string, birthdate: string }	{}	Kein Fehler: 201 Daten ungültig: 400 Serverfehler: 500

Softwarearchitektur Backend



Das Backend beinhaltet wie bereits erwähnt sechs mit dem Framework „Express“ erstellte REST Funktionen. Mit diesen Funktionen werden die Daten in der JSON-DB gespeichert, manipuliert, gelöscht und gelesen.

Softwarearchitektur Frontend

Das Frontend wird durch das Framework „Next.js“ verwaltet. Dieses ermöglicht es, mittels Ordnerstruktur Routing für die Website festzulegen, wodurch ein klarer und übersichtlicher Aufbau des Projektes entsteht. Trotzdem gibt es eine Komponente, in der alle anderen Komponenten aufgerufen werden. Diese ist in der Datei „_app.js“ zu finden.

In dieser von „Next.js“ generierten Datei wurden zwei Modifikationen durchgeführt. Zum einen wurde hier der „SessionProvider“ eingebettet und zum anderen wurde ein Kontext definiert, über den die Komponenten „NotificationBar“ und „Modal“ kontrolliert werden. Diesen „OverlayContext“ kann man von überall im Projekt aufrufen um ein Modal, bzw. eine Notificationbar mit einem individuellen Inhalt zu öffnen.

Die Komponenten „Modal“ und „NotificationBar“, sowie alle Komponenten, die keine selbstständigen Seiten sind, wurden in einem Ordner namens „components“ definiert. Dieser „components“ Ordner hat zusätzlich eine „index.js“ Datei, in der die Exporte gebündelt werden, was das Importieren vereinfacht.

Es gibt außerdem eine Klasse namens „RequestHandler“, in der eine private Funktion namens „#genericFetch“ und sechs weitere öffentliche Funktionen definiert sind. Die öffentlichen Funktionen rufen die „#genericFetch“ Funktion mit unterschiedlichen Eingabeparametern auf. Dadurch wird der Aufwand zum Einbinden von Endpunkten minimiert.

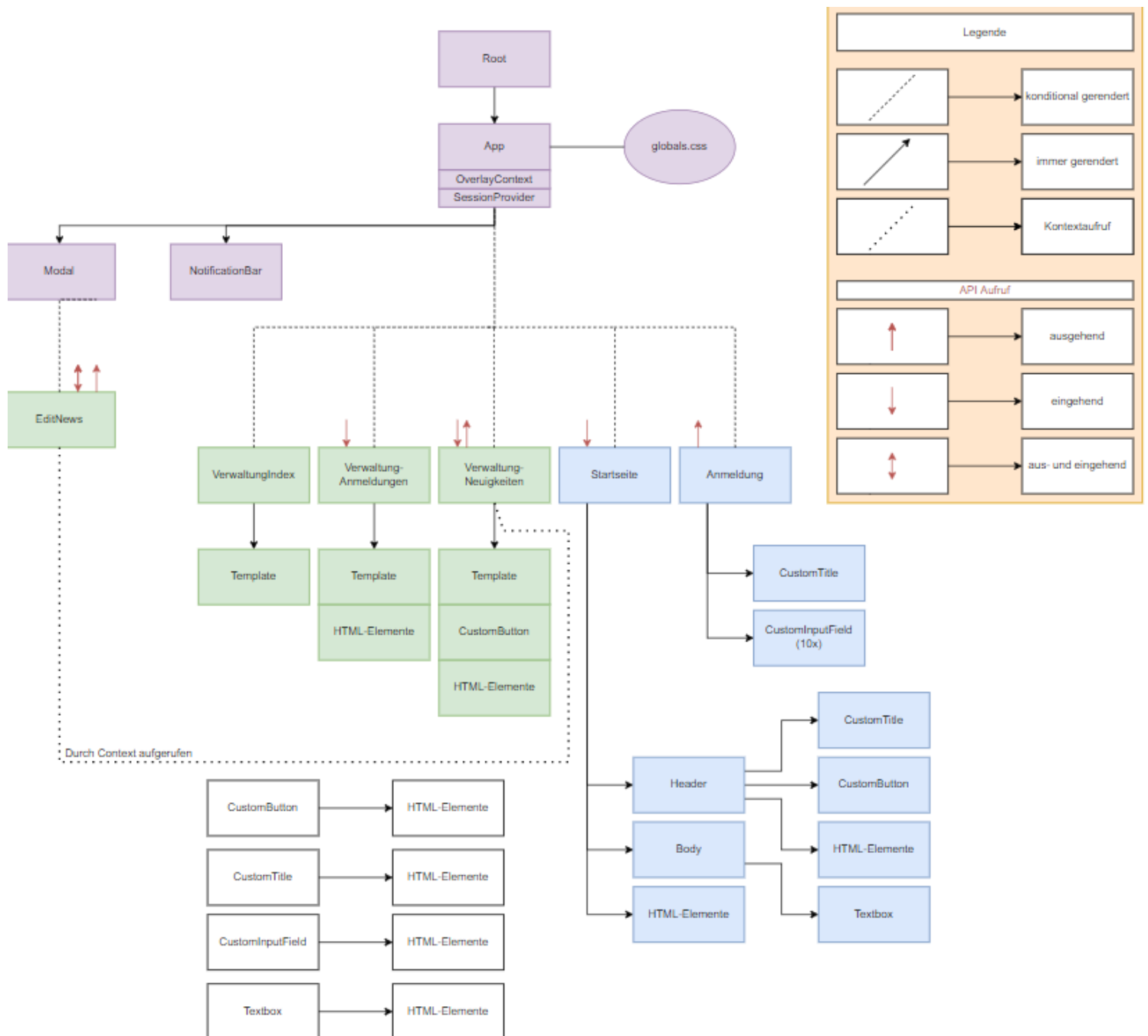


Abbildung 18: Vollständiges Architektordiagramm Frontend

Third Party Software

Um alle nachfolgenden Frameworks und Softwares zu installieren, wurde der „Node Package Manager“ („npm“) verwendet. Außerdem haben wir „Git“ in „GitHub“ als Versionskontrollsystem benutzt.

Zum Erstellen dieses Projektes wurden einige Frameworks verwendet. Um die Endpunkte für die API zu erstellen wurde das Framework „Express“ verwendet. Werden diese Endpunkte aufgerufen, wird die Library

„node-json-db“ verwendet. Mit ihr kann man einfach Daten in lokalen JSON-Dateien abspeichern. Außerdem wurde „Express“ noch um Middleware Libraries erweitert. „Express“ wird mit „node“ gestartet.

Für das Frontend wurden das Framework „Next.js“ in Kombination mit der Library „React“ verwendet. React ist eine JavaScript Library um User Interfaces zu bauen. „Next.js“ baut auf „React“ auf und erweitert es um Funktionalität. Um die Komponenten zu gestalten, wurde das CSS-Framework „tailwindcss“ verwendet. Hier kann man von „tailwindcss“ definierte Attribute in die Klassennamen der HTML-Tags schreiben, wodurch ihnen CSS Attribute zugewiesen werden. Für die Icons wurde die Library „tabler-icons-react“ verwendet.³

Arbeitszeitaufteilung

Zu Beginn der Projektarbeit haben wir zusammen als Gruppe die Vorgehensweise besprochen und die grundlegende Struktur der Anwendung geplant. In dieser Phase sind unter anderem die Use Cases entstanden. Daraufhin hat jeder einen Designvorschlag für die Benutzeroberfläche entwickelt, die dann als Team selektiert und verfeinert wurden. Nun haben wir uns in zwei dreier Teams aufgeteilt und je einen Teil der Designs umgesetzt. Da wir das CSS-Framework „Tailwind“ benutzt haben, hat Cedric, der hiermit bereits Erfahrung hat, dem Rest der Gruppe eine kurze Einführung gegeben. Bei der Umsetzung des Projekts in JavaScript und der Einbindung von externen Frameworks haben die zwei erfahrensten Teammitglieder Cedric und Sebastian den Rest der Gruppe unterstützt, teils im Pair-Programming oder durch Beantwortung von Fragen. Die Dokumentation und Präsentation wurden wiederum vom ganzen Team erarbeitet. Meist wurde in kurzen Intervallen selbstständig gearbeitet und die Ergebnisse zu festen Terminen in der großen Gruppe zusammengetragen.

³ Siehe Literaturverzeichnis unter Software

Literaturverzeichnis

Software

- Git <https://git-scm.com/>
- GitHub <https://github.com/>
- Node <https://nodejs.org/>
- Npm <https://www.npmjs.com/>
- React <https://reactjs.org/>
- NextJS <https://nextjs.org/>
- Express <https://expressjs.com/>
- node-json-db <https://www.npmjs.com/package/node-json-db>
- tabler-icons-react <https://www.npmjs.com/package/tabler-icons-react>
- cors <https://www.npmjs.com/package/cors>
- body-parser <https://www.npmjs.com/package/body-parser>
- unqiud <https://www.npmjs.com/package/unqiud>
- next-auth <https://www.npmjs.com/package/next-auth>
- express-fileupload <https://www.npmjs.com/package/express-fileupload>

Andere

- [1] REST Standard von Microsoft: <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design#define-api-operations-in-terms-of-http-methods>
- [2] React Context: <https://reactjs.org/docs/context.html>