Hochschule Aalen

Praxissemesterbericht
Studiengang : Informatik

Teamkalender Web-App

von

Gabriel Roth

82798

Betreuender Professor: Sebastian Stigler

Einreichungsdatum : 28.Februar 2024

Eidesstattliche Erklärung

Hiermit erkläre ich, **Gabriel Roth**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

Als Bestandteil meines Studiums im Studiengang Software Engineering an der Hochschule Aalen ist ein Projekt vorgesehen. Als Projektthema habe ich mich für eine Teamkalender Web-App entschieden. Zum Thema Teamkalender bin ich gekommen, weil ich einen Kalender haben wollte, an dem sich jedes Familienmitglied beteiligen kann. Da ich bereits aus dem Praxissemester Erfahrungen mit Angular gesammelt habe, habe ich mich dazu entschieden hierfür auch Angular zu nutzen.

In der heutigen digitalen Zeit nehmen Web Applikationen eine wachsende Rolle ein, seit meines absolvierten Prakikums sind Web-Apps auch für mich von großer Bedeutung. Im Laufe der Projektarbeit wird verstärkt auf die Nutzung eines geeigneten Verifizierungsverfahrens eingegangen. Als Verifizierungsverfahren wurde Keycloak genutzt, dabei wird auf die Features, die Keycloak mit sich liefert und welche Vor- und Nachteile Keycloak mit sich bringt im Vergleich mit anderen möglichen Verifizierungsverfahren.

Da die Teamkalender Web-App im Netzwerk verfügbar ist und für Personen in unterschiedlichen Altersklassen genutzt werden soll, wurde auch insbesondere auf eine benutzerfreundliche Bedienbarkeit der Web-App geachtet, sodass auch Personen ohne IT-Kenntnisse diese Teamkalender Web-App problemlos nutzen können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Mit der Teamkalender Web-App soll es ermöglicht werden einen Kalender, auf mehreren Endgeräten, in bspw. einer Familie führen zu können, in dem die Termine aller Personen angezeigt werden. Bei den Terminen soll zwischen privat und öffentlich unterschieden werden. Öffentliche Termine sind für alle Personen sichtbar und private Termine sind nur für den Benutzer sichtbar, der sie auch selbst erstellt hat.

Die Teamkalender Applikation hat den Mehrwert, dass die Nutzer einsehen können, wann sich ein Termin für alle Personen zusammen am Besten eignet. Vorausgesetzt keine Person hat zu diesem Zeitpunkt einen privaten Termin.

1.2 Problemstellung und -abgrenzung

Problemstellung:

- Geeignetes Authentifizierungsverfahren nutzen
- Benutzerfreundliche Gesaltung

Problemabgrezung:

- Backend der Datenbank
- Backend des Authentifizierungsverfahrens
- Frontend

1.3 Ziel der Arbeit

Es soll eine Teamkalender Web-Applikation mit Angular erstellt werden, welche für mehrere Personen gleichzeitig, auf unterschiedlichen Endgeräten nutzbar sein soll. Dabei soll ein Authentifizierungsverfahren genutzt werden. Die Web-App soll für

Jedermann genutzt werden können, auch für Personen ohne Informatikkenntnisse, deshalb sollte die Nutzung der Web-App so einfach wie möglich gestaltet sein, so wie eine übersichtliche und benutzerfreundliche Darstellung im Frontend bieten.

1.4 Vorgehen

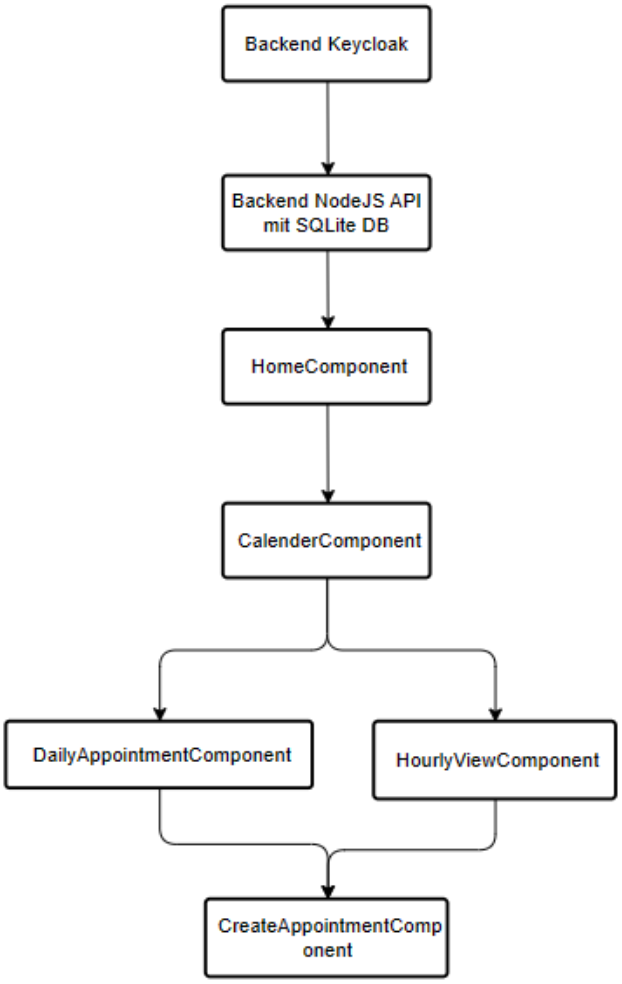


Abbildung 1.1: Fluss Diagramm

2 Grundlagen

2.1 Axure RP 10

2.1.1 Planung des Frontends

Für die vorab Planung des Designs der Teamkalender Kalender Web-App wurde Axure RP 10 genutzt. Mit Axure RP 10 kann man Prototypen für Web-Applikationen, Apps und andere Software mit einem GUI erstellen. Dies ist auch schnell und einfach handzuhaben [2].

Mit Axure RP 10 kann man Prototypen erstellen. Diese können verschiedene Seiten umfassen und somit auch schon das Routing, das für den gesamten Projektumfang notwendig ist erstellen. Mithilfe von Interactions kann die Weiterleitung von einer Seite auf die andere ermöglicht werden. Für Prototypen könnnen Libraries, das sind bspw. Widgets wie Schalflächen, Bilder, Texte und Formen genutzten werden. Neben den Libraries können auch Komponenten direkt in Axure erstellt werden, wie bspw. ein Navigation Header. Buttons oder andere Widgets können auch durch Style direkt in Axure nach Größe und Farbe angepasst werden. Style bringt den Vorteil, dass durch das Exportieren von Axure nachher nicht mehr viel mit CSS angepasst werden muss, da man es so wie zuvor im Prototyp konfiguriert auch weiterhin nutzen kann [1].

2.1.2 Vorschau und Exportieren von HTML und css

Durch den Play Button kann eine Vorschau im Webbrowser geöffnet werden und dadurch kann das Routing mit dem Prototyp selbst getestet werden. Dadurch können auch vorzeitig Fehler entdeckt und sofort in Axure behoben werden.

Axure kann den Prototypen konvertiert mit HTML-, Css-, JavaScript- und Bilddateien zusammen als eine HTML-Ausgabe exportieren. Sodass diese auch in Projekten weiter genutzt und weiter verarbeitet werden können [3].

2.2 Angular

Angular ist ein großes umfangreiches Frontend-Framework für Single-Page Applikationen. Grundsätzlich basiert es auf Typescript. Angular läuft Clientseitig und wird meist mit Backends, welche eine REST API enthalten, verknüpft. Dadurch können effiziente und nutzerfreundliche Web-Apps gebaut werden. Mithilfe von Angular lassen sich dynamische Apps und User-Interfaces einfach erstellen, dennoch bietet Angular auch viele erweitere Features, wie bspw. einen eingebauten HTTP-Client oder RxJS. Die Verwendung von auf TypeScrpit basierten Features ist optional. Auch reines JavaScript kann verwendet werden [9].

2.2.1 Command Line Interface

Angular hat sein eigenes Command Line Interface (CLI). Durch das neue Projekte, so wie Komponenten und Services angelegt werden können. Wenn Node auf dem System installiert ist, dann besteht auch die Möglichkeit einen lokalen Dev-Server zu starten, da der Node Package Manager (npm) verwendet wird um Angular zu installieren [9].

2.2.2 Ordnerstruktur

Nach dem erstellen der App erstellt Angular neben Soruce-Files auch Dateien zum tracken der npm-Abhängigkeiten mithilfe der 'package.json'. Angular spezifische Konfigurationen werden in der Datei ängular.json"gespeichert. TypeScript Konfigurationen werden in der Datei 'tsconfig.json' gespeichert. Im Ordner '/src/app' sind alle Komponenten und Services gespeichert. Im 'src' Ordner existiert auch eine Datei 'index.html' diese ist der Ausgangspunkt der Angular web-App. In der ïndex.html"werden die Root-Komponenten durch den tag <app-rout> eingebunden. Von dieser Stelle aus werden nun alle Komponenten geschachtelt.

Der Einstiegspunkt ist die 'main.ts', in ihr werden die 'AppModules' geladen. Diese Datei ist gleichzeitig auch noch der zentrale Punkt der Web-App da, über die "main.ts"Module und Services importiert werden [9].

2.2.3 Komponenten

Eine Komponente besteht aus einem Template (HTML), der Logik (Typescript) und den Styles (CSS order SCSS). Ein Decorator wird zu Beginn benötigt, in diesem werden die Rahmenbedingungen, so wie der Pfad zum HTML und CSS für die Komponente festgelegt. Danach folgt eine relativ normale Klassendefinition, in

dieser werden dann Methoden und Attribute definiert. Die Methode "ngOnInit()ïst eine sogenannte Lifecycle-Methode, das heißt, wenn eine Funktion darin ausgeführt wird, dann wird sie genau dann ausgeführt, wenn die Komponente initiiert wird [9].

2.2.4 Attribute und Interpolation

Um Attibute dynamisch in das HTML-Template einbinden zu können, das wären in dem Fall Variablen aus dem Typescript, welche man im HTML anzeigen möchte wird die SString-Interpolation"genutzt. Diese wird im HTML durch die 2 geschweiften Klammern "<VariablenName>"deutlich gemacht. Dadurch wird dann im Browser die Value der Variablen angezeigt [9].

2.2.5 Inputs

Inputs werden benötigt, um Daten in eine Komponente übergeben zu können. Das sieht so aus '"[]- Input' und nennt sich Databinding bzw. Propertybinding [9].

2.2.6 Outputs und Events

'"()- Output' bei Outputs handelt es sich um Eventbinding. Dabei wird in den Klammern das Event beschrieben, auf das reagiert werden soll, bspw. ein 'click' event. Darauf folgt eine Methode aus dem Typescript der Komponente, welche dann durch die Ausführung des Events ausgeführt werden sollte [9].

2.2.7 Directives

Directives sind Logik-Bausteine, welche teilweise mit Angular ausgeliefert werden, aber auch selbst definiert werden können. Durch diese kann das Aussehen, bzw. das Verhalten von Elementen im Document Object Model (DOM) verändert werden. Directives können in Elemente eingebaut und mit diesen verwendet werden. Mithilfe von 'ngClass' kann einem Element dynamisch eine spezielle Klasse zugewiesen werden, um bspw. spezielle CSS-Regeln auf das Element anwenden zu können. 'ngFor' ist eine Art For-Schleife, mit der eine Komponente für z.B. mehrere Elemente eines Arrays erzeugt werden kann, das heißt das wäre dann eine 'Strukturelle Directive'. Eine weitere 'strukturelle Directive' ist 'ngIf'. Dies ist wie eine gewöhnliche if-else Struktur anzusehen, also werden dadurch gewisse Elemente zu einer Bedingung angezeigt [9].

2.2.8 Services

Services werden strikt von Komponenten getrennt, dies ist notwendig, um die Wiederverwertbarkeit und die Modularität des Codes zu verbessern. Daduch werden die Komponenten um einiges aufgeräumter, da man die Funktionalitäten die direkt etwas mit der Ansicht macht, von anderen Prozessen, quasi im Hintergrund trennt. Somit können dann spezielle Aufgaben an verschiedene Services von einer Komponente deligiert werden. Ein Service wird nach dem folgendermaßen benannt <name>.service.ts [9].

2.2.9 Observables und Subscriptions

Damit Angular Clientseitig weiterhin funktionsfähig bleibt, während es auf eine Antwort auf einen Request wartet werden von Angular 'Observables' zur Verfügung gestellt. Ein 'Observable' ist ein Objekt, welches beobachtet werden kann und dementsprechend auch Rückmeldungen weitergeben kann, sobald diese eintreffen. Anhand folgendes Beispiels wird es nun besser erläutert:

```
1 dummyMethod(): Observable<string>{
2         const data = of("dummy");
3         return data;
4     }
```

Quelltext 2.1: Observables Beispiel

An diesem Beispiel kann man nun gut erkennen, dass die dummyMethod() ein Observable liefert. Dies nennt sich auch ein Observable abbonieren, sodass es durchgängig überwacht wird. Dies würde würde in einer Komponente so aussehen:

```
1 this.dummyService.dummyMethod().subscribe(
2             (data) => (this.dummy_atribute = data);
3         );
```

Quelltext 2.2: subscirbe Beispiel

Hier wird das abbonieren durch die subscribe()-Funktion dargestellt. Dabei wird innerhalb der subscribe()-Funktion eine Arrow-Function durch den '=>' angegegeben. Durch den '=>' wird dann veranschaut, was mit den übergebenen Daten passieren soll [9].

2.2.10 Subjects und Subscriptions

Subjects sind eine andere Art von Subscriptions, es ist ein einzelnes Element. Ein Subject kann auch als Quell-Observable bezeichnet werden, da es auch mehrere

Ziele geben kann, die durch das Subject beobachtet werden. Mehrere Ziele gäbe es genau dann, wenn es auch mehre Subscriptions gibt [9].

2.2.11 HTTP Client

Der HTTP Client muss in der app.module.ts importiert und dem imports Array hinzugefügt werden, damit er für alle Komponenten als import zur Verfügung steht. Er wird benötigt, um bspw. GET/POST Requests an eine API senden zu können [9].

2.2.12 Forms

Durch 'ngModel' kann ein 'Two-way Databinding' erzeugt werden. Das heißt ein Attribut der Komponente wird mit einem Input Element beidseitig verknüpft. Dies wird durch '("[()]- Input + Output)' ermöglicht [9].

2.2.13 Routing

Das Routing passiert im Hintergrund, dadurch bleibt es bei einer Single-Page Web Applikation. Das RouterModule muss in der app.module.ts importiert und dem imports Array hinzugefügt werden, damit es für alle Komponenten als import zur Verfügung steht [9].

Das Routing selbst wird ebenfalls in der app.module.ts vorgenommen

```
1 const routes: Routes = [
2   { path: '', component: HomeComponent, canActivate:
      [KeycloakGuard] },
3   { path: 'daily-appointment', component:
      DailyAppointmentComponent, canActivate: [KeycloakGuard] }
4 ]
```

Quelltext 2.3: Routing Beispiel

Nun kann man einen Pfad mit einer Komponente verknüpfen, welche durch diesen Pfad erreichbar ist. In der app.component.html muss nur noch der Tag '<router-outlet>' hinzugefügt werden, damit die Routes auch genutzt werden können [9].

2.3 Aufbau der Architektur der Web-Applikation

2.3.1 Backend Node.js API mit SQLite Datenbank

Node.js basiert auf der V8-JavaScript-Engine und ist eine plattfromübergreifende Open-Source-Laufzeitumgebung und Open-Source-Bibliothek. Es wird zum Ausführen von Webanwendungen außerhalb des Browser Clients verwendet. Das heißt also für die serverseitige Progrmmarieung. Node.js wird hauptsächlich für Backend API Dienste, nicht bblockierende, erignisgesteuerte Server wie Websites verwendet. In meinem Fall für die Backebnd API mit SQLite Datenbank [7].

2.3.2 Backend Keycloak

Um die WebApp abzusichern wurde ein Single Sign on Mechanismus benötigt. Dafür habe ich die Open Source Library Keycloak Angular verwendet.

2.3.3 Frontend

Das Frontend ist in seperate Komponenten aufgeteilt, so dass diese übersichtlich und einfach zu verarbeiten sind. Clientseitiges Routing, um Seiten einfach wechseln zu können, ohne die Seite neu laden zu müssen. Benutzerfreundliche Gestaltung und einfache Bedienbarkeit der Web-App.

2.4 Keycloak

2.4.1 Library Keycloak Angular

Die Library vereinfacht die Integration vom Keycloak Single Sign on in eine Angular App. Sie bietet mehrere Funktionen.

Der Keycloak Service, der die in Angular verwendeten Methoden umschließt durch 'keycloak-js'. Dieser gibt den ursprünglichen Funktionen zusätzliche Funktionen und fügt neue Funktionen hinzu, um die Nutzung von Angular Applikationen zu erleichtern.

Eine generische AuthGuard Implementierung, sodass die AuthGuard-Logik angepasst werden kann, indem die Authentifizierungslogik geerbt wird. Ein HttpClient Interceptor, der allen HttpClient Anfragen den Autorisierungsheader hinzufügt. Damit ist es auch möglich das Routing vom Authorisierungsheader asuzuschließen,

also den Zugriff auf bestimmte Seiten gewährleisten, nur dann wenn der Benutzer auch authorisiert ist [6].

2.4.2 Keycloak Administrator Client

Mit dem Keycloak Administrator Client kann das Backend konfiguriert werden. Darin können viele verschiedene Einstellungen vorgenommen werden. Es können darin realms konfiguriert werden. Jedes realm verwaltet Benutzer und enthält mehrere CLients. Für die Authentifizierung können darin auch Einstellungen, wie bspw. Registrierung neuer Benutzer mit Emailauthentifizierung etc., wenn ein Benutzer sein Passwort vergessen hat kann der Admin darin für den Benutzer ein neues Passwort zuweisen [5].

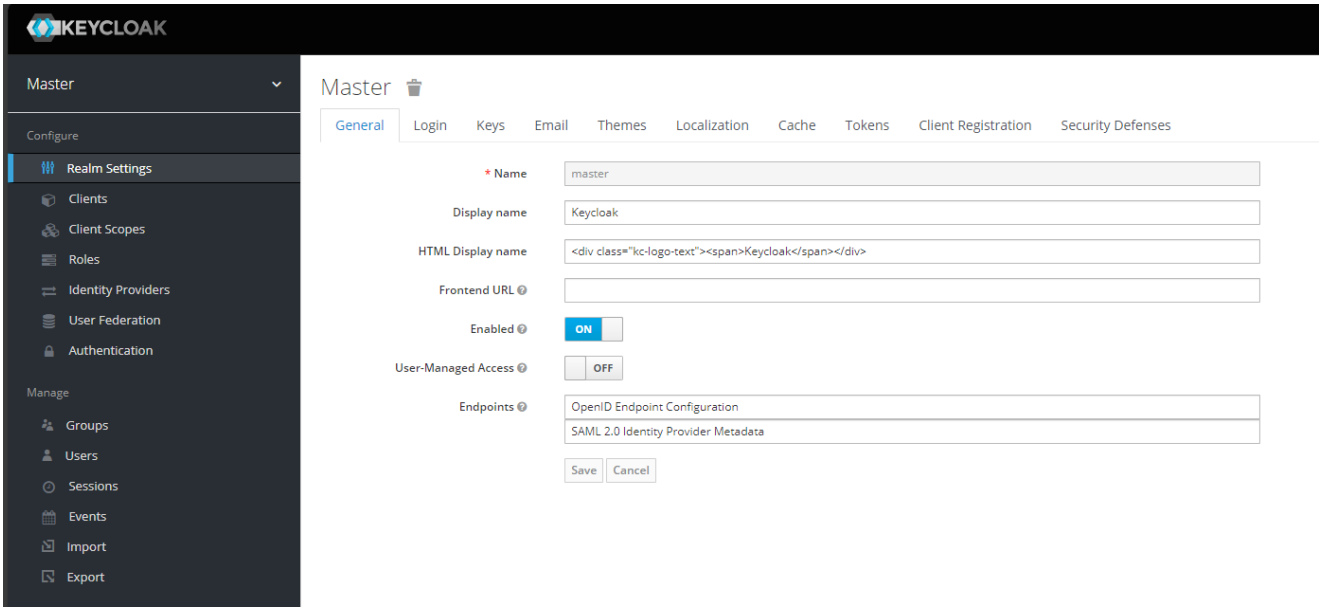


Abbildung 2.1: Keycloak Administrationsoberfläche

2.5 Server

Der Server ist notwendig, um die Web-App auf mehreren Endgeräten gleichzeitig nutzen zu können und dabei ist auch nur eine Inbetriebnahme auf dem Server notwendig.

2.5.1 BWcload

Die bwcloud ist eine 'Infrastructure as a Service' Umgebung, welche speziell für die Forschung und Lehre in Baden-Württemberg entwickelt wurde und betrieben wird. Mit bwCloud können virtuelle Maschinen oder virtuelle Server erzeugt, gestartet und betrieben werden. Virtuelle Maschinen unterscheiden sich in der Administration und im Betrieb nicht von echten physischen Maschinen. Jede virtuelle Maschinen hat die folgenden Ressourcen zur Verfügung: virtuelle CPUs, RAM, IPv4/IPv6 und Speicherplatz. Die Größe der virtuellen Maschinen ist regionsabhängig. Mithilfe der bwCloud können je nach Bedarf einzelne oder mehrere VMs gestartet werden und dabei in unterschiedlichen Regionen betrieben werden [4].

2.6 Postman

Postman ist eine API-Platform zum Erstellen und Verwenden von APIs. Mit Postman wieder jeder Schritt des API-Lebenszyklus vereinfacht. Postman bietet dabei auch API-Repositories. Es verfügt auch über verschiedene Werkzeuge für die Beschleunigung des API-Lebenszyklus, dies betrifft das Design, Tests, Dokumentation und Mocking bis zur Freigabe und Auffindbarkeit von APIs [8].

3 Problemanalyse

Die Teamkalender Web-App ist in ein Frontend und zwei Backends unterteilt. Das Frontend ist üblicherweise die GUI für den Benutzer. Das zweite Backend ist eine NodeJS API mit einer SQLite Datenbank. Das zweite Backend wird für Keycloak benötigt.

Frontend:
Das Frontend beinhaltet Benutzerregistrierung, bzw. Benutzeranmeldung und die Funktionalitäten der Teamkalender App.

Dadurch, dass die Web-Applikation abgesichert sein soll, muss die Registrierung, bzw. Anmeldung im Voraus erfolgen.

Die Benutzerregistrierung bzw. Benutzeranmeldung wird von Keycloak übernommen.

Die Funktionalitäten der Teamkalender-Web App lässt sich unterteilen in:

- Kalenderlogik
- Stundenanzeige der Termine
- Termine erstellen
- Unterscheidung zwischen öffentlichen und privaten Terminen
- Priorisierung von Terminen

Backend der NodeJS API mit SQLite Datenbank:
Das erste Backend ist eine NodeJS API mit einer Datenbank. Durch die API soll die Schnittstelle für den Datenaustausch zwischen Frontend und Backend ermöglicht werden. Als Datenbank wird SQLite genutzt, in dieser sollen alle Termine gespeichert werden, in dem Fall mit dem Inhalt, des vom Benutzer selbst erstellten Termins.

Backend von Keycloak:
In diesem werden die Benutzerdaten gespeichert, die für die Registrierung, bzw. die Anmeldung erforderlich sind. Neben den Benutzerdaten werden in diesem Backend auch die Realm, Client und die zusätzlichen Benutzereinstellungen, welche Keycloak in einem eigenenen Administrations Frontend bietet, gespeichert.

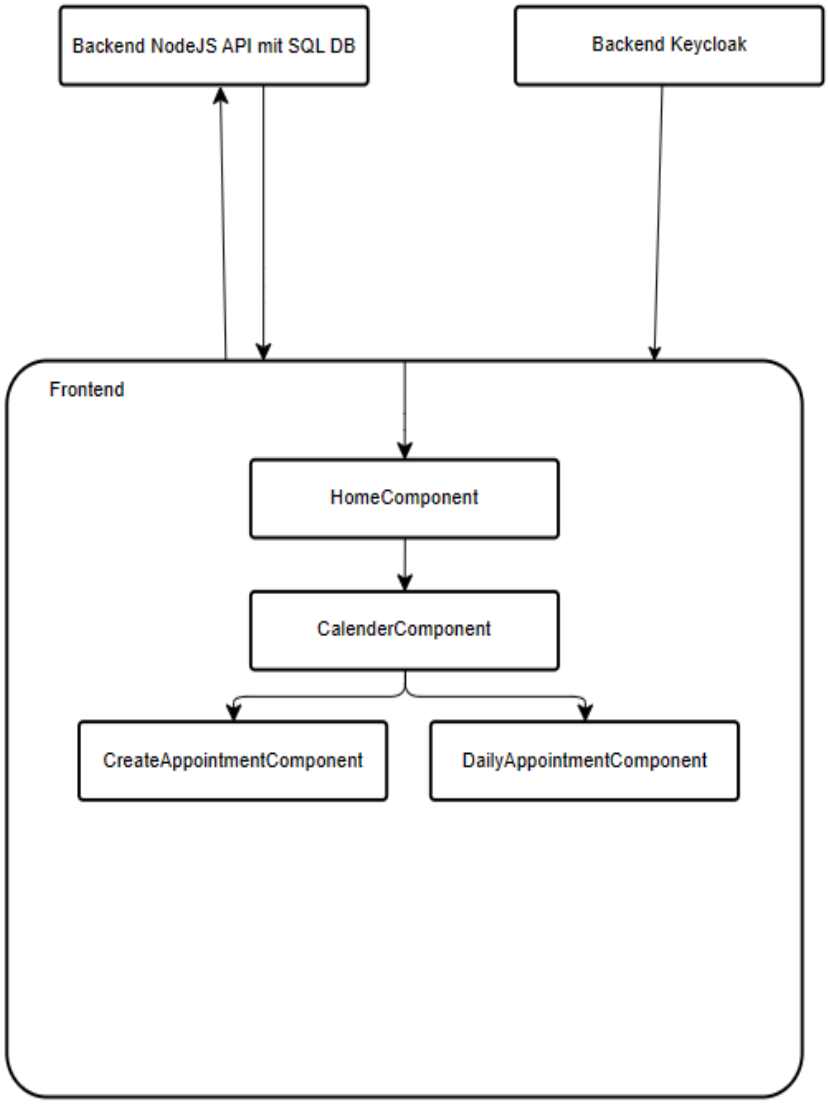


Abbildung 3.1: Aufbau

chapterEntwurf labelcha:Entwurf

Für das Frontend wurde vor der Implementierung mithile von Axure RP 10 ein Prototyp erstellt.

Für die vorab Planung, vor der Implementierung wure das Design der Teamkalender Kalender Web-App mithilfe von Axure RP 10 erstellt. Es wurde dabei speziell darauf geachtet, dass das Design, das Routing und die Funktionalitäten dadurch enthalten sind. Aus Axure RP 10 wurde dann HTML und das dazugehörige CSS exportiert. CSS und HTML wurde letztendlich nur teilweise beibehalten, da es von Axure RP 10 einfach nur irgendwie gebaut wurde, ohne dabei auf die kleinste Art und Weise auf die Namensgebung der CSS Klassen or IDs zu achten.

Ordnerstruktur des Frontends:

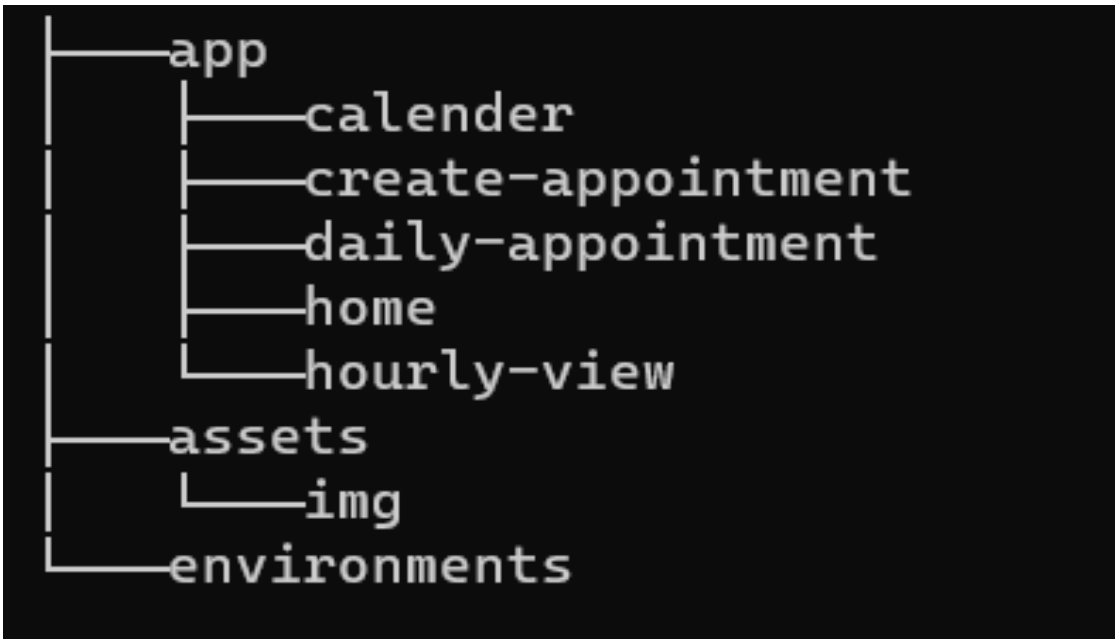


Abbildung 3.2: Ordnerstruktur Frontend

Kommunikation untereinander:

Kommunikation ziwschen Frontend und Backend von Keycloak:

Für die Kommunikation werden Docker Container genutzt, das heißt das Backend von Keycloak läuft in einem eigenen Docker Container und das Frontend läuft in einem eigenen Docker Container. Dabei wird dann über die zugehörige IP und den Port der jeweiligen Docker Container kommuniziert.

Kommunikation zwischen Frotend und Backend der NodeJS API mit SQLite DB:

Abbildung 3.3: Kommunikation Frontend und Backend Keycloak

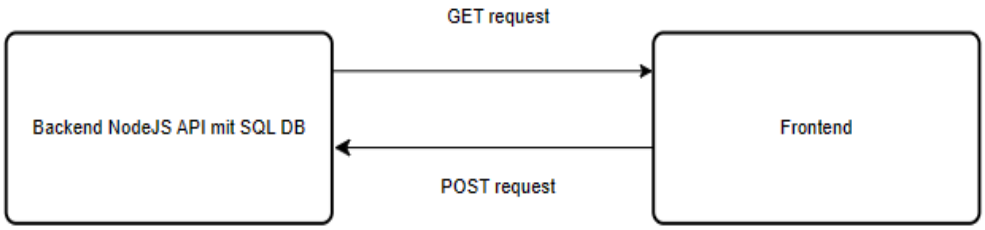


Abbildung 3.4: Kommunikation Frontend und Backend NodeJS API mit SQLite DB

Das Backend der NodeJS API mit SQLite Datenbank läuft über node, das heißt hierfür wird kein Docker Container benötigt, da über Endpunkte mithilfe von Post und Get Requests mit dem Frontend kommuniziert wird.

Inbetriebnahme: Muss auf einem Server sein, da sonst die App nur im Localhost läuft und nicht den Sinn und Zweck erfüllt, dass sie von mehreren Person gleichzeitig genutzt werden kann.

4 Implementierung und Entwicklung

Für die Implementierung des Frontends wurde Angular verwendet. Demnach besteht jede Komponente aus HTML, CSS, Specification Typescript und Typescript.

Für die Implementierung des Backends wurde nodeJS verwendet.

Frontend:

Anfangs wurde aus Axure RP 10 der darin erstellte Prototyp in HTML exportiert, um ihn als Grundlage für HTML und CSS zu nutzen.

Anschließend wurde eine Hello World Implementierung durchgeführt und die Routes im Routing Module implementiert. Dieser Schritt war notwendig um einerseits das Routing testen zu können und nachdem auch Keycloak in Betrieb genommen wurde den abgesicherten Bereich seitens Keycloak einzurichten.

```
1 const routes: Routes = [
2   { path: '', component: HomeComponent, canActivate:
      [KeycloakGuard] },
3   { path: 'daily-appointment', component:
      DailyAppointmentComponent, canActivate: [KeycloakGuard] },
4   { path: 'create-appointment', component:
      CreateAppointmentComponent, canActivate: [KeycloakGuard] },
5   { path: 'calender', component: CalenderComponent, canActivate:
      [KeycloakGuard] },
6
7   // home
8   { path: '**', redirectTo: '' }
9 ];
```

Quelltext 4.1: AppRoutingModule

Quelltext 4.1 stellt die möglichen Routes dar. Die abgesicherten Routing-Bereiche von Keycloak sind durch 'canActivate: [KeycloakGuard]' erkennbar. Das heißt diese Routen sind nur zugänglich, wenn der Benutzer auch über Keycloak authentifiziert ist, welche durch den KeycloakGuard (siehe Keycloak) überprüft werden.

Um den abgesicherten Bereich zu erkennnen wurde der AuthGard von Keycloak benötigt [6].

```
1  public async isAccessAllowed(
2      route: ActivatedRouteSnapshot,
3      state: RouterStateSnapshot
4    ) {
5      if (!this.authenticated) {
6        await this.keycloak.login({
7          redirectUri: window.location.origin + state.url
8        });
9      }
10
11     const requiredRoles = route.data['roles'];
12
13     if (!Array.isArray(requiredRoles) || requiredRoles.length ===
           0) {
14       return true;
15     }
16
17     return requiredRoles.every((role) =>
           this.roles.includes(role));
18   }
```

Quelltext 4.2: Keycloak.Guard.ts

Wenn der Benutzer nicht authentifiziert ist wird er erneut aufgefordert sich anzumelden (Quelltext 4.2, Zeile 5-9). Es wird die Rolle des Users überprüft, die für die Route benötigt wird(Quelltext 4.2, Zeile 11). Wenn keine spezielle Rolle erfordert wird, dann soll die Route erreicht werden können(Quelltext 4.2, Zeile 13). Wenn die Rolle des Benutzers übereinstimmt, dann soll die Route erreichbar sein(Quelltext 4.2, Zeile 17-18).

Dann wurde die Open Source Library Keycloak Angular eingebunden und in app.module.ts wurde durch die initializeKeycloak()-Funktion Keycloak initialisiert [6].

```
1  function initializeKeycloak(keycloak: KeycloakService) {
2    return () =>
3      keycloak.init({
4        config: {
5          url: 'http://193.197.231.167:8080/auth', //bwcloud IP can
               be changed!
6          realm: 'teamkalender-realm',
7          clientId: 'teamkalender_client'
8        },
9        initOptions: {
10         redirectUri: 'http://193.197.231.167:4200', //bwcloud IP
               can be changed!
11         checkLoginIframe: false
12       },
13       loadUserProfileAtStartUp: true
14     });
15 }
```

Quelltext 4.3: initializeKeycloak() AppModule

Anschließend wurde für Keycloak ein Dockerfile erstellt, mit dem ein Docker Image gebaut wurde und dieses dann als Docker Container gestartet werden konnte.

Mit diesem Docker Container wurde dann getestet, ob es möglich ist diesen über ein Windows-Subsystem (wsl) mit Ubuntu-20.04 laufen zu lassen, sodass dieser einer IP zugeordnet ist und gleichzeitig das Frontend über Port Forwarding von Github zu nutzen. Dies war allerdings erfolglos, da das Portforwarding von Github über https läuft, wodurch ein Konflikt mit dem Docker Container entsteht, da dieser über http läuft.

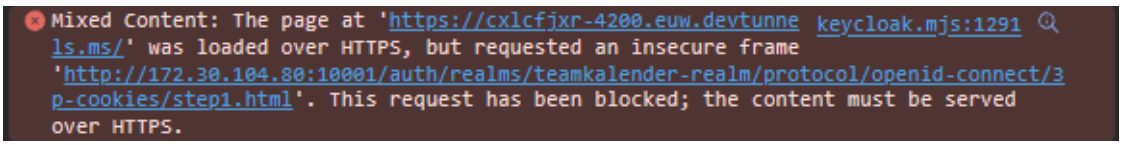


Abbildung 4.1: Fehlermeldung

Dadurch wurde dann festgestellt, dass eine virtuelle Maschine, ein virtueller Server, ein Raspberry Pi oder ähnliches benötigt wird, sodass das Frontend und das Backend unter einer IP laufen.

Danach wurde das Backend der NodeJS API mit einer SQLite Datenbank implementiert.

```
app.post('/events', (req, res) => {
    const { date, teamEvent, startTime, endTime, priority,
        matter, comment, owner } = req.body;
    if (!date || !teamEvent || !startTime || !endTime ||
        !priority || !matter || !comment || !owner) {
        return res.status(400).json({
            status: 'error',
            error: 'All fields are required',
        });
    }
    const sql = "INSERT INTO events(date, teamEvent, startTime,
        endTime, priority, matter, comment, owner) VALUES
        (?,?,?,?,?,?,?,?)";
    db.run(sql, [ date, teamEvent, startTime, endTime, priority,
        matter, comment, owner], (err) => {
        if (err) {
            return res.status(500).json({ status: 500, success:
                false, error: err.message });
        }
        console.log('Successful input',  date, teamEvent,
            startTime, endTime, priority, matter, comment, owner);
        res.status(200).json({ status: 200, success: true });
    });
});
```

Quelltext 4.4: POST-Request NodeJS API

Mit dem POST request werden alle im Frontend eingegeben Daten in der Datenbank gespeichert. Dabei wird geprüft, ob alle Daten vorhanden sind, ansonsten wird ein Error zurückgegeben und die Daten werden nicht in der Datenbank gespeichert.

```
app.get('/events', (req, res) => {
    const sql = "SELECT * FROM events";

    db.all(sql, [], (err, rows) => {
        if (err) {
            return res.status(500).json({ status: 500, success:
                false, error: err.message });
        }

        res.status(200).json({ status: 200, success: true, data:
            rows });
    });
});
```

Quelltext 4.5: GET-Request NodeJS API

Mit dem GET request werden alle Daten aus der Datenbank zurückgegeben.
Die POST und GET Requests wurden mithilfe von Postman getetstet (siehe Test).

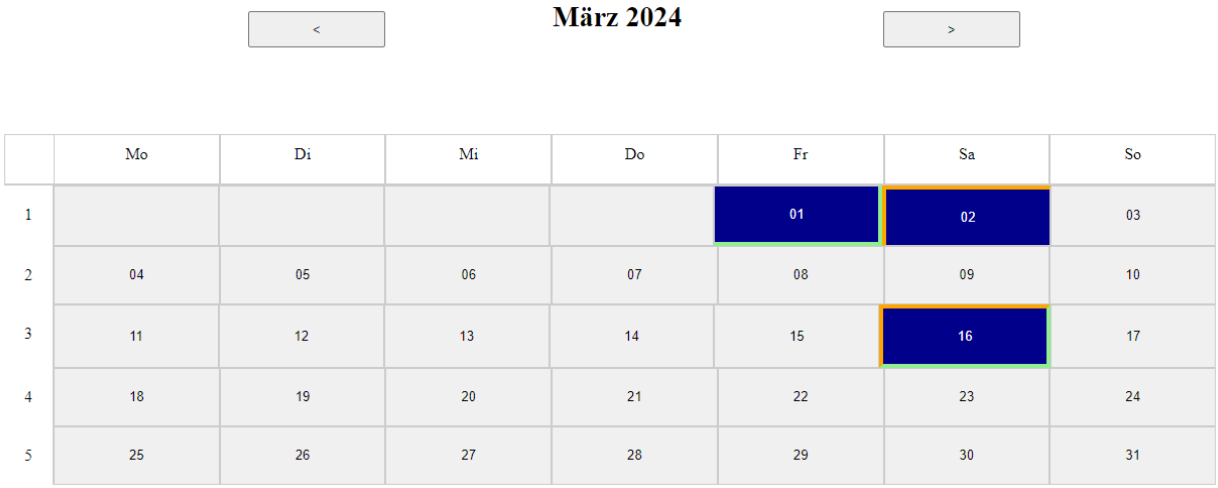
Das create Table für die Datenbank ist folgendermaßen aufgebaut:

```
const createTableQuery = `
    CREATE TABLE IF NOT EXISTS events (
        ID INTEGER PRIMARY KEY,
        date TEXT,
        teamEvent BOOLEAN,
        startTime TEXT,
        endTime TEXT,
        priority INTEGER,
        matter TEXT,
        comment TEXT,
        owner TEXT
    )
`;
```

Quelltext 4.6: create Table SQL-Datenkbank Backend

Dabei erhält jeder erstellte Termin eine ID, das gewählte Datum, die Unterscheidung ob öffentlich/privater Termin, die Startzeit für den Terminbeginn, die Endzeit für das Ende des Termin, die Priorität des Termins, der Grund bzw. der Name des Termins, den Kommentar und der Besitzer. Nachdem die Datenbank mithilfe von Postman ausgiebig getestet wurde, kam es auch schon zur Kalenderlogik.

Für die Kalenderlogik gab es verschiedene Möglichkeiten, da es optisch viele verschiede Designs für Kalender gibt und dahinter jeweils eine andere Logik steckt. Ich habe mich an dem Design des Standard Windows Kalenders orientiert.



	Mo	Di	Mi	Do	Fr	Sa	So
1					01	02	03
2	04	05	06	07	08	09	10
3	11	12	13	14	15	16	17
4	18	19	20	21	22	23	24
5	25	26	27	28	29	30	31

Abbildung 4.2: Kalender

Für die Anordnung der Tage ist die generateDays()-Methode zuständig.

```
1   generateDays(year: number, month: number) {
2     const daysInMonth = new Date(year, month, 0).getDate();
3     const firstDayOfMonth = new Date(year, month - 1, 1).getDay();
4     const offset = firstDayOfMonth === 0 ? 6 : firstDayOfMonth -
          1; // offset for the first week until the first day of the
          month is reached, first day is a Sunday if firstDayOfMonth
          is 0
5     const daysToReachSunday = daysInMonth + (7 - ((daysInMonth +
          offset) % 7));// Total days until the next Sunday after
          the current month days is reached
6     this.days = [];
7
8     for (let i = 1 - offset; i <= daysToReachSunday; i++) {
9       if (i <= 0 || i > daysInMonth) {
10        this.days.push(null); // push empty day fields for the
              first and last week of the month
11      } else {
12        this.days.push(new Date(year, month - 1, i));
13      }
14    }
15
16    const lastWeek = this.days.slice(-7); //creates a new Array
          with the last 7 elements of this.days
17    const lastWeekHasData = lastWeek.some(day => day !== null);
          // checks if at least one day isn't empty
```

```
18
19      if (!lastWeekHasData) {
20        this.days = this.days.slice(0, -7); // remove last week if
              it's empty
21      }
22
23      this.weekNumbers = this.getWeekNumbers(year, month);
24    }
```

Quelltext 4.7: generateDays()-Methode CalenderComponent

Mit der generateDays()-Methode wird ein Array erstellt, in dem die Tage für den jeweiligen Monat zusammen mit den für die Struktur des Kalenders benötigten leeren Feldern am Anfang und am Ende des Monats. Als Übergabeparameter benötigt diese Funktion das Jahr und den Monat. Da aus diesen dann die Gesamtzahl an Tagen des jeweiligen Monats (Quelltext 4.6, Zeile 2) und in die Konstante daysInMonth geschrieben werden. Die Postition ersten Tages vom jeweiligen Monat wird in die konstante Variable firstDayOfMonth geschrieben (Quelltext 4.6, Zeile 3). In die Konstante offset werden die Tage, bis der erste Tag des Monats in der ersten Woche erreicht ist gezählt. Falls der erste Tag ein Sonntag ist wird 6 zurückgegeben, sonst firstDayOfMonth - 1 (Quelltext 4.6, Zeile 4). In die Konstante daysToReachSunday wird die Gesamtzahl an Tagen, bis der nächste Sonntag erreicht ist, geschrieben (Quelltext 4.6, Zeile 5). In der for-Schleife wird von der ersten Stelle im Kalender, deshalb 1 - offset bis zur Konstante daysToReachSunday hochgezählt und dem entsprechend für die leeren Kalendertage, am Anfang des Kalenders, vor dem Beginn des Monats und am Ende des Kalenders, nach dem Monat, leere Tage durch null in das Array gepusht (Quelltext 4.6, Zeile 8-15). Die Konstante lastWeek enthält ein Array mit der im Kalender sichtbaren letzten Woche, das heßißt auch Tage, die nicht mehr zum aktuellen Monat gehören, jedoch Bestandteil der letzten Woche die im Kalender sichtbar ist (Quelltext 4.6, Zeile 16). Dann wird überprüft, ob mindestens einer der Tage noch zum aktuellen Monat gehört und dementsprechend true oder false in in die Konstante lastWeekHasData geschrieben(Quelltext 4.6, Zeile 17). Falls die letzte Woche keinen Tag des aktuellen Monats enthält soll diese entfernt werden(Quelltext 4.6, Zeile 19-21).

Für die Darstellung der Kalenderwochen eines Monats im Kalender selbst ist die chunkedDays()-Methode notwendig.

```
1   get chunkedDays(): (Date | null)[][] {
2     const chunkSize = 7;
3     const arrayCopy = this.days.slice();
4     const chunks = [];
5     while (arrayCopy.length > 0) {
6       const chunk = arrayCopy.splice(0, chunkSize); //removes
            chunkSize from index 0
7       chunks.push(chunk);
8     }
9     return chunks;
10  }
```

Quelltext 4.8: get chunkedDay()-Methode CalenderComponent

In der ChunkedDays()-Funktion werden die Tage der jeweiligen Monate in Wochen aufgeteilt. Dafür wird eine Kopie des Arrays this.day genutzt und die Konstante arrayCopy geschrieben (Quelltext 4.7, Zeile 3). Für jeden Schleifen-Durchlauf wird ein Teil aus dem Array arrayCopy entfernt und in die Konstante chunk geschrieben und durch chunkSize in Teile von 7 Elemeneten, in dem Fall in einzelne Wochen unterteilt. Anschließend wurden sind in das Array chunks gepusht(Quelltext 4.7, Z.4-7). Im Array chunks werden dann am Ende die grupppierten Teile des Array chunks zurückgegeben (Quelltext 4.7, Z.9).

Um die Termine im Kalender passend anzeigen lassen zu können wurde die hasEventsOnDay()-Methode genutzt.

```
1
2   hasEventsOnDay(day: Date | null, teamEvent: number): boolean {
3     if (day instanceof Date) {
4       const year = day.getFullYear();
5       const month = (day.getMonth() + 1).toString().padStart(2,
            '0');
6       const date = day.getDate().toString().padStart(2, '0');
7       const dayStr = `${date}.${month}.${year}`;
8
9       return this.events.some((event: any) => {
10        if (event.date && event.teamEvent === teamEvent) {
11          let isCurrentUserEvent: boolean;
12          if (teamEvent === 1) {
13              isCurrentUserEvent = this.isCurrentUser(event);
14          } else {
15              isCurrentUserEvent = true;
16          }
17          return event.date === dayStr && isCurrentUserEvent;
18        }
19        return false;
20      });
21    }
22    return false;
23  }
```

Quelltext 4.9: hasEventsOnDay()-Methode CalenderComponent

In der hasEventsOnDay()-Methode wird das Datum zuerst mithilfe eines Stringbuilders in einen String, dass die deutsche Datum Syntax (TT.MM.YYYY), enthält umgewandelt (Quelltext 4.8, Zeile 4-7). Dies ist erforderlich, um es nachher mit event.date vergleichen zu können, da dieses schon in der deutschen Datum Syntax ist. Wenn mindestens ein Termin existiert, dann wird überprüft überprüft, ob der Termin einem Datum zugeordnet ist und verglichen ob es sich um einen privaten oder öffentlichen Termin handelt, mit der Variablen teamevent (Quelltext 4.8, Zeile 10) und es wird true zurückgegeben, ansonsten false. Dann wird überprüft, ob teamevent === 1 ist, also ob es sich hierbei um einen privaten Termin handelt. Ist dies der Fall wird geprüft, ob der Termin auch dem aktuellen Benutzer gehhört (Quelltext 4.8, Zeile 12-16). Es wird true zurückgegeben, wenn das Datum von event.date mit dayStr übereinstimmt und es sich gleichzeitig um ein privates Event des aktuellen Benutzers bzw. um öffentliches Event handelt, ansonsten wird false zurückgegeben und das Event wird für den aktuellen Beneutzer nicht angezeigt.(Quelltext 4.8, Zeile 17-19).

In der darin genutzten isCurrentUser()-Methode wird geprüft, ob der Besitzer des Termins dem aktuell eingeloggten Benutzer entspricht.

Die onDayClick()-Funktion ist für die Weiterleitung des Benutters in der GUI notwendig.

```
1   onDayClick(day: Date) {
2     if (day instanceof Date) {
3       this.router.navigate(['/daily-appointment'], { queryParams:
            { date: day.toISOString() } });
4     }
5   }
```

Quelltext 4.10: onDayClick()-Methode CalenderComponent

Bei der Weiterleitung wird der Tag als ISOString per query mit weitergegeben (Quelltext 4.9, Zeile 3).

Um das ganze im Frontend anzeigen lassen zu können, wurde folgender HTML Code genutzt.

```
1         <div class="week-labels">
2             <div></div>
3             <div>Mo</div>
4             <div>Di</div>
5             <div>Mi</div>
6             <div>Do</div>
7             <div>Fr</div>
8             <div>Sa</div>
9             <div>So</div>
10        </div>
11        <div class="calendar-days">
12            <ng-container *ngFor="let chunk of chunkedDays; let i
                  = index">
13                <div class="calendar-row">
14                    <div class="week-numbers">{{ [i + 1] }}</div>
15                    <button *ngFor="let day of chunk"
                          class="calendar-day"
16                        [routerLink]="day ?
                              ['/daily-appointment', day] : null"
                              (click)="day ? onDayClick(day) : null"
17                        [ngClass]="{ 'has-events':
                              hasEventsOnDay(day,1) ||
                              hasEventsOnDay(day,2),
                              'public-border':hasEventsOnDay(day,2),
                              'private-border':hasEventsOnDay(day,1)}">
18                        {{ (day | date: 'dd') }}
19                    </button>
20                </div>
21            </ng-container>
22        </div>
```

Quelltext 4.11: Calender.component.html

Mit ngFor muss jedes Element des 2 dimensionalen Arrays chunkedDays durchlaufen werden, welches mit der chunkedDays Methode erzeugt wurde (Quelltext 4.10, Zeile 12). Let i=index wird für das Anzeigen der Kalenderwochen für den aktuellen Monat benötigt (Quelltext 4.10, Zeile 14). Die Buttons werden für jedes Element des in der ngFor durchlaufenen Arrays chunk angezeigt (Quelltext 4.10, Zeile 15-19). Der routerLink und das Anklicken des Tages ist nur dann möglich, wenn es auch ein Tag ist, der im Kalender angezeigt wird, ansonsten kann man leere Tage nicht anklicken, da für diese auch keine Funktionen vorgesehen sind. Wird auf einen Tag geklickt, der auch im Kalender angezeigt wird, dann wird man durch den routerLink auf die Seite der täglichen Termine weitergeleitet und die durch die Angular (click)-Funktion wird der angeklickte Tag als ISO weitergeleitet, mithilfe der onDayClick()-Funktion. Durch die Angular Funktion ngClass werden dann die borders und die Farbe des Buttons, wenn für den Tag auch Termine vorgesehen sind, angepasst. Dies geschieht in css dadurch, dass die Css Class has-events nur dann verwendet wird, wenn die Funktion hasEventsOnDay(day, 1) oder hasEventsOnDay(day, 2) aufgerufen wird, dann wird der Button für den jeweiligen Tag blau markiert, ansonsten nicht (Quelltext 4.10, Zeile 17.1-17.3). Die Border der Buttons werden links und oben orange, durch die css Klasse public-border, wenn es für den Tag öffentliche Termine gibt, das heißt die Funktion wird folgendermaßen so hasEventsOnDay(day, 2) aufgerufen. Die Border wird rechts und unten hellgrün, durch die css Klasse private-border, wenn die Funktion hasEventsOnDay(day,1) aufgerufen wird (Quelltext 4.10, Zeile 17.4-17.5). Für den jeweiligen Button wird im Frontend dann der Tag angezeigt (Quelltext 4.10, Zeile 18).

Im CSS sind die css-Klassen so aufgebaut.

```
1  .has-events {
2    background-color: darkblue;
3    color: white;
4  }
5  .public-border {
6    border-top: 4px solid orange;
7    border-left: 4px solid orange;
8  }
9
10 .private-border {
11   border-bottom: 4px solid lightgreen;
12   border-right: 4px solid lightgreen;
13 }
```

Quelltext 4.12: Calender.component.scss

Nach dem Kalender folgt die dazugehörige Stundenansicht, in der dailyAppointment-Component. Da die Stundenansicht allerdings nicht nur in der dailyAppointment-Component sichtbar sein sollte, sondern auch in der HomeComponent, wurde die HourlayAppViewComponent ausgelagert. Dadurch das die Komponente ausgelagert ist kann sie in beliebigen Komponenten im HTML einfach aufgerufen werden.
In der Hourly-View werden die Termine für den entsprechend davor in der CalenderComponent ausgewählten Tag angezeigt.

Die Termine sollen in der Hourly-View ähnlich wie in der TagesAnsicht angezeigt werden, mit dem Unterschied, dass in der Hourly-View die Termine stündlich angezeigt werden und den genau den Bereich abdecken, für den Termin auch vorgesehen ist. Die Termine sind nach Prioritäten markiert, Hohe Priorität ist dem entsprechend rot markiert, mittlere Priorität Gelb und niedrige Priorität grün. Die Unterscheidung zwischen öffentlichen und privaten Terminen ist hier auch nochmal deutlicher zu sehen, da für jeden Termin die komplette Border entsprechend für öffentliche Termine orange wird und für private Termine grün.

Die Prioritäten werden anhand der vom Benutzer ausgewählten Prioriät in der CreateAppointmentComponent festgelegt.

```
1   getPriorityClass(priority: number): string {
2     switch (priority) {
3       case 1:
4         return 'priority-green';
5       case 2:
6         return 'priority-yellow';
7       case 3:
8         return 'priority-red';
9       default:
10        return '';
11    }
12  }
```

Quelltext 4.13: hourly-view.component.ts

Abhänig von der im Termin festgelegten Priorität wird dann, ähnlich wie der Priorität die entstprechende background-color den CSS-Klassen priority-green, priority-yellow und priority-red zugeordnet und davon dann immer genau eine aktiv, wenn der Termin eine Priorität hat.

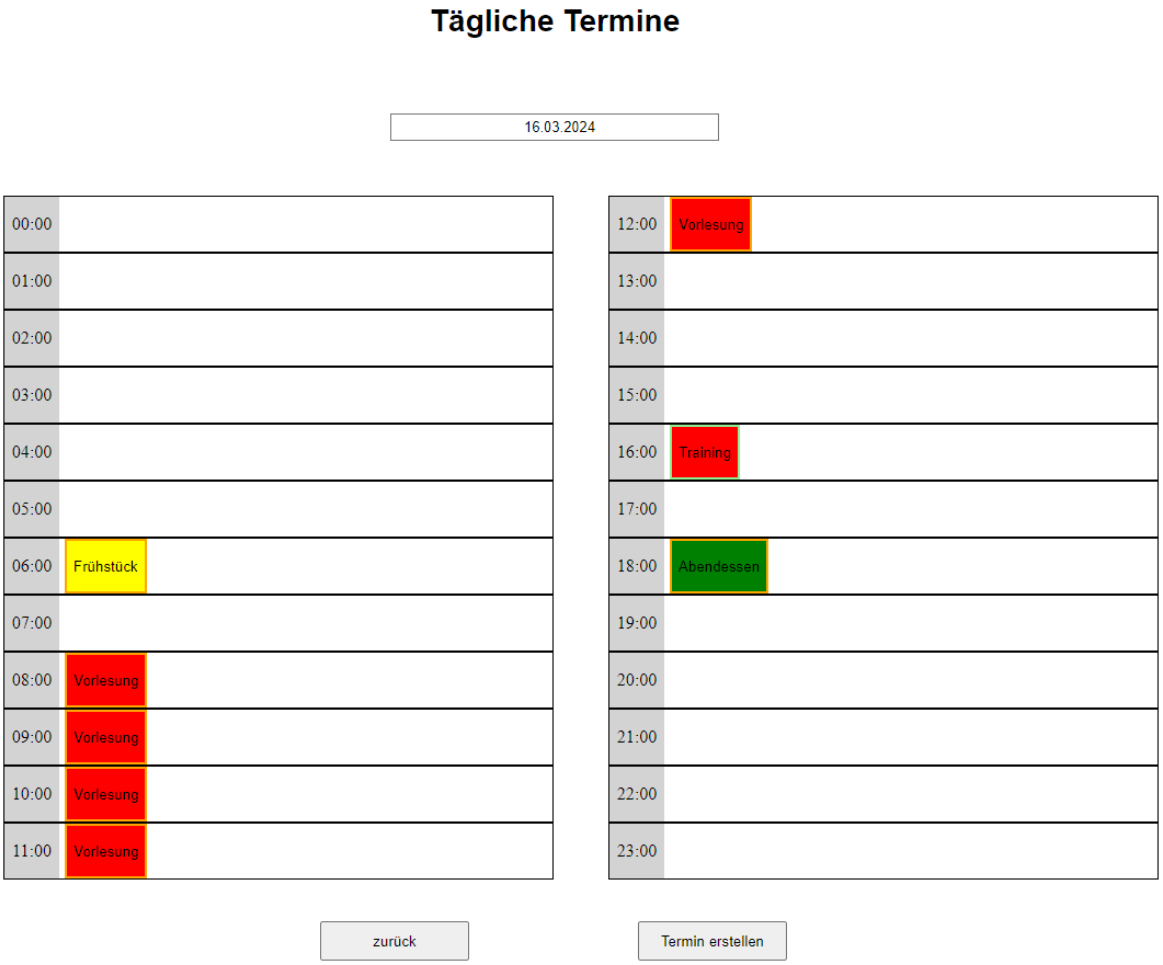


Abbildung 4.3: Tägliche Termine

Die getPriorityClass()-Funktion wird in der isTeamEvent()-Methode aufgerufen, da in dieser Funktion neben der Änderung der background-color im CSS auch noch die Border, je nachdem ob der Termin öffentlich oder privat ist orange oder grün gefärbt.

```
1
2   isTeamEvent(priority: number, teamEvent: number): string {
3     let classes = this.getPriorityClass(priority);
4
5     if (teamEvent === 1) {
6       classes += ' private-border';
7     } else if (teamEvent === 2) {
8       classes += ' public-border';
9     }
10    return classes;
11  }
```

Quelltext 4.14: hourly-view.component.ts

Der isTeamEvent()-Funktion werden als Übergabeparameter die Variablen priorirty und teamEvent übergeben. Erst wird der classes Variablen die zurückgegebene CSS Klasse der getPriorityClass()-Funktion übergeben (Quelltext 4.13, Z.3) und anschließend wird auch die CSS Klasse von private-border, wenn es sich um einen privaten Termin handelt und somit in teamEvent 1 steht, angehangen. Wenn es sich um einen öffentlichen Termin handelt, also teamEvent 2 ist, dann wird CSS Klasse public-border an die Variable classes angehangen und am Ende wird die Variable classes zurückgegeben (Quelltext 4.13, Z.5-10). Somit hat dann jeder Termin in der Stundenansicht einen farbigen Hintergrund je nach Priorität und eine farbige Border je nachdem ob privat oder öffentlich.

Um die Stunden, die für den Termin festgelegt sind auf der Stundenansicht verteilt anzeigen zu lassen wird die isHourMarked()-Funktion benötigt.

```
1   isHourMarked(appointment: any, hour: string): boolean {
2     return hour >= appointment.startTime && hour <
          appointment.endTime;
3   }
```

Quelltext 4.15: hourly-view.component.ts

Die isHourMarked()-Funktion bekommt als Übergabeparameter den Termin durch die Variable appointment und die Stunden in der Variablen hour (Quelltext 4.14, Zeile 1). Die Funktion gibt dann entsprechend die Stunden ab Beginn des Termins mit der Variablen appointment.startTime bis zum Ende des Termins mit der Variablen appointment.endTime zurück(Quelltext 4.14, Zeile 2-2.1).

Klickt man auf einen Termin wird entsprechend der Termin im Detail mithilfe eines PopUp-Fensters angezeigt.

```
1
2   public openpopup(appointment: any) {
3     appointment.show = true;
4   }
5
6   public closepopup(appointment: any) {
7     appointment.show = false;
8   }
```

Quelltext 4.16: hourly-view.component.ts

Durch die Funktionen openpopup() und closepopup() wird das PopUp-Fenster durch appointment.show = true angezeigt und durch das 'X' im PopUp-Fenster wird das PopUp-Fenster wieder geschloßen, durch appointment.show = false. Die Funktion wird im HTML aufgerufen, sobald auf einen Termin geklickt wird.

Der HTML Teil der hourViewComponent sieht so aus.

```
1  <div id="base" class="">
2    <div id="hourlyView-pos">
3      <div class="hourly-view">
4        <div class="hour-column" *ngFor="let column of [0, 1]">
5          <ng-container *ngFor="let hour of hours; let i = index">
6            <div class="hour-row"
7              *ngIf="(column === 0 && i < hours.length / 2) ||
                   (column === 1 && i >= hours.length / 2)">
8              <div class="hour">
9                <p>{{ hour }}</p>
10             </div>
11             <div class="appointment" *ngFor="let appointment of
                  appointments">
12               <span
13                 *ngIf="isHourMarked(appointment, hour) &&
                       (appointment.teamEvent === 2 ||
                       (appointment.teamEvent === 1 &&
                       isCurrentUser(appointment)))">
14                 <div class="wrapper">
15                   <button id="appointmentButton"
                         (click)="openpopup(appointment)"
16                       [ngClass]="isTeamEvent(appointment.priority,
                           appointment.teamEvent)">{{
                           appointment.matter
17                       }}</button>
18                 </div>
19                 <div class="overlay" *ngIf="appointment.show">
20                   <div class="content">
```

```
21                      <div class="close"
                            (click)="closepopup(appointment)">
22                        &times;
23                      </div>
24                      <p>Datum: {{appointment.date}}</p>
25                      <p>Termin Thema: {{appointment.matter}}</p>
26                      <p>Kommentar: {{appointment.comment}}</p>
27                      <p>Beginn:{{appointment.startTime}}</p>
28                      <p>Ende: {{appointment.endTime}}</p>
29                      <p>Ersteller: {{appointment.owner}}</p>
30                    </div>
31                  </div>
32                </span>
33              </div>
34            </div>
35          </ng-container>
36        </div>
37      </div>
38    </div>
39  </div>
```

Quelltext 4.17: hourly-view.component.html

Hierbei wurde auch wie bei der dem Kalender schon ziemlich viel mit css gearbeitet. Die ngFor ist dafür da den Stundenansicht in 2 gleichgroße Spalten aufzuteilen, in dem Fall Spalte 0 und Spalte 1 (Quelltext 4.16, Zeile 4). In der 2. ngFor werden die Stunden hochgezählt (Quelltext 4.16, Zeile 5). Diese werden in der ngIF genutzt und damit für Spalte 0 die Hälfte der Gesamtzahl an Stunden, in dem Fall 00:00 bis 11:00 für die erste Spalte gentutzt und für die zweite Spalte wird der die größere Hälfte der Stunden gentuzt, in dem Fall 12:00 bis 23:00 (Quelltext 4.16, Zeile 7-7.1). Darin werden dann die Stunden angezeigt(Quelltext 4.16, Zeile 8-10). Durch die ngFor wird dann für jeden Termin im Array Appointments (Quelltext 4.16, Zeile 11-11.1), durch die ngIf geschaut, ob es sich um einen öffentlichen Termin handelt, durch appointment.teamEvent === 2 oder ob es sich um einen privaten Termin handelt und der Besitzer des Termins der aktuell eingeloggte Benutzer ist(Quelltext 4.16, Zeile 12-13.3). Trifft einer der beiden Fälle zu, dann werden die Termine in der Farbe ihrer Priorität und der dazugehörigen privaten oder öffentlichen Border angezeigt, mit der Variablen appointment.matter im Vordergrund für den Benutzer sichtbar, das wäre in dem Fall der Termin Grund bzw. der Termin Name (Quelltext 4.16, Zeile 15-18). In der ngIf wird geprüft ob auf den Button mit dem jeweiligen Termin geklickt wurde, ist dies der Fall soll das Pop-Up Fenster geöffnet werden(Quelltext 4.16, Zeile 19), in dem dann im Detail der Termin angezeigt wird. Durch css wird dann der Hintergrund schwarz abgedeckt und im Vordergrund ein weißes div erzeugt mit dem Inhalt: Datum: appointment.date Termin Thema: appointment.matter Kommentar: appointment.comment Beginn:appointment.startTime Ende: appointment.endTime Ersteller: appointment.owner (Quelltext 4.16, Zeile 24-30).

Durch &times wird dann das ’X’ erzeugt, durch dass das PopUp-Fenster wieder geschloßen werden kann, mithilfe der closepopup()-Funktion(Quelltext 4.16, Zeile 21-23).

Das PopUp-Fenster sieht dann durch CSS folgendermaßen aus.

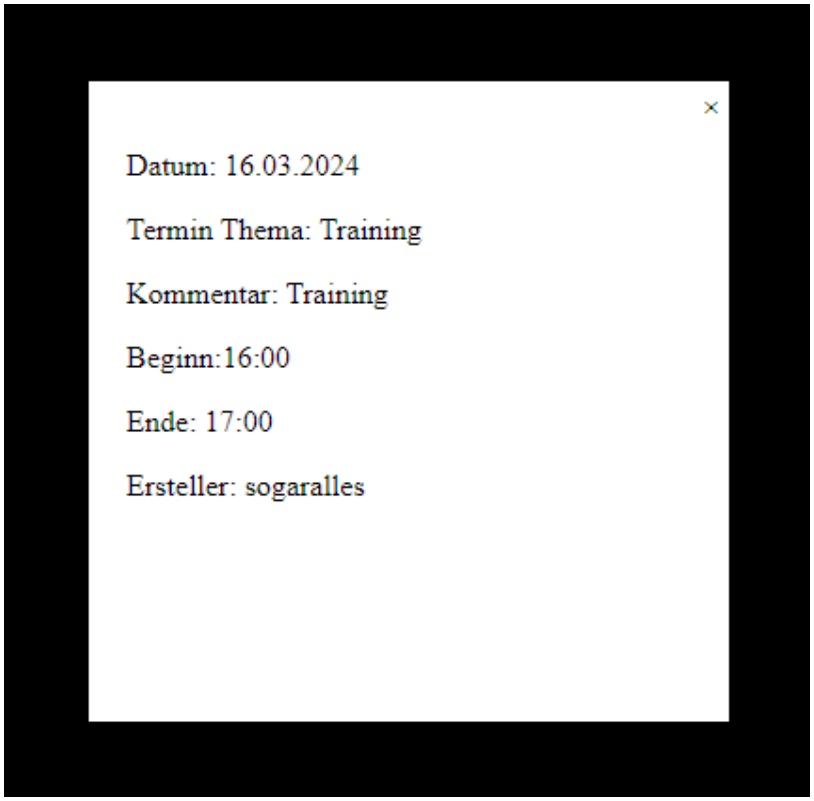


Abbildung 4.4: Pop-Up

Dafür wurde mit CSS einmal das komplette Fenster schwarz bedeckt, dann ein kleineres content Fenster, mit weißem Hintergrund, mittig platziert und oben rechts im weißen content Fenster ein 'X' platziert, mit welchem das PopUp-Fenster wieder geschloßen werden kann.

```
1
2   .content {
3     background-color: white;
4     width: 300px;
5     height: 300px;
6     padding: 20px;
7     position: relative;
8     z-index: 1;
9   }
10
11
12  .overlay {
13    position: fixed;
14    top: 0;
15    left: 0;
16    background-color: black;
17    width: 100%;
18    height: 100%;
19    display: flex;
20    justify-content: center;
21    align-items: center;
22    z-index: 1;
23  }
24
25  .close {
26    position: absolute;
27    top: 5px;
28    right: 5px;
29    cursor: pointer;
30  }
```

Quelltext 4.18: hourly-view.component.scss

Die CSS Klasse content wird benötigt, um das eigentliche PopUp-Fenster mit weißem Hintergrund zu erzeugen und durch z-index: 1 gegenüber dem anderen Inhalt der Seite hervor zu heben. Die Css Klasse overlay verdeckt das komplette Fenster schwarz und hebt sich durch z-indedx: 1 hervor. Und die CSS Kalsse close ist für das 'X' oben rechts im Container der Content Klasse, um das PopUp-Fenster wieder zu schließen.

Um Termine erstellen zu können wurde die CreateAppointmentComponent benötigt.

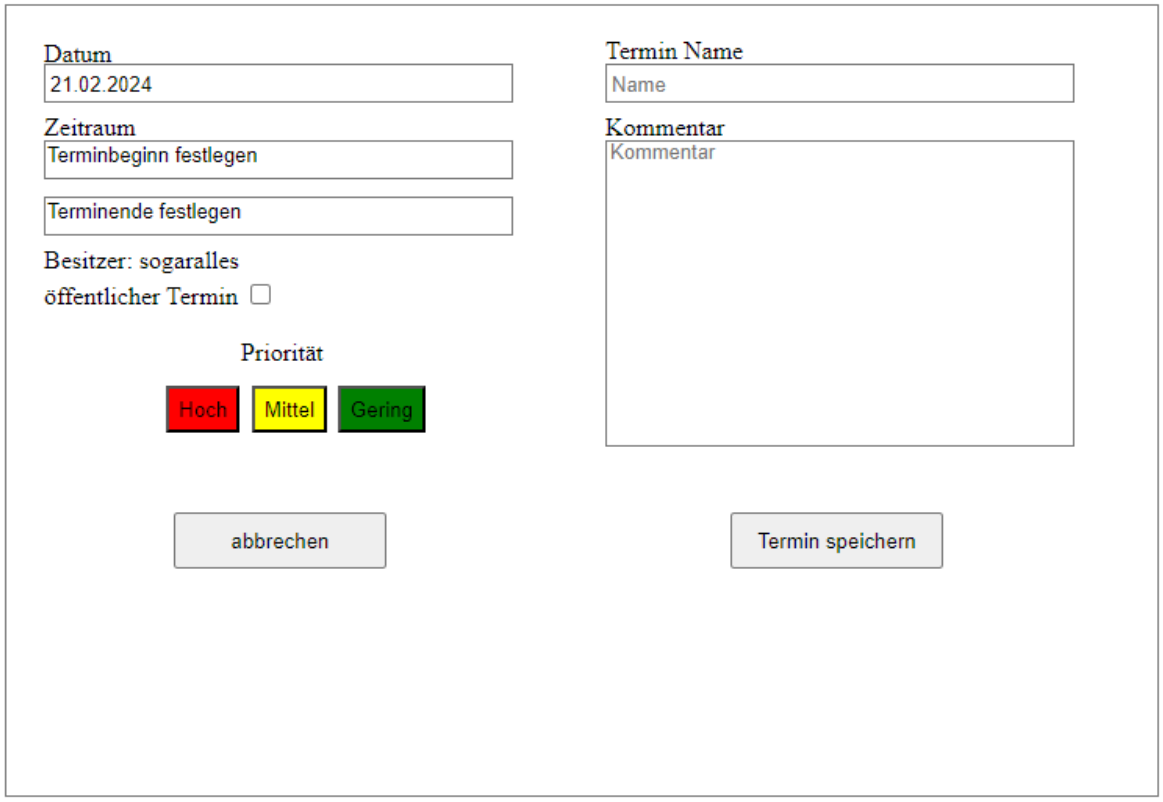


Abbildung 4.5: Termin erstellen

In dieser musste auch zuerst wieder das als ISO8601 übergebene Datum der HourlyAppViewComponent mithilfe eines Stingbuilders in die deutsche Syntax gebracht werden, da es sonst im Frontend für den Benutzer nicht korrekt angezeigt wurde. Erst war auch geplant nur die ISO8601 zu nutzen, allerdings gab es dabei Probleme mit der Zeitzone, wodurch es nicht möglich war nur die ISO8601 zu nutzen. Denn dadurch das nur die ISO8601 genutzt wurde waren dann die Tage im Kalender, nachdem sie erstellt wurden alle um einen Tag versetzt, sodass die Termine im Kalender nicht korrekt dargestellt wurde und auch nicht das richtige Datum zugeordnet bekommen haben.

Die Startzeit und die Endzeit wird mithilfe 2 Dropdown Menüs festgelegt.

```
1   toggleDropDown(dropDownMenu: string, hour: string) {
2     this.selectedHour = hour;
3
4     if (dropDownMenu === 'dropDownStartTime') {
5       this.startTimeValue = this.selectedHour;
6       this.isDropDownStartTime = !this.isDropDownStartTime;
7       this.isDropDownEndTime = false;
8     } else if (dropDownMenu === 'dropDownEndTime') {
9       this.endTimeValue = this.selectedHour;
10      this.isDropDownEndTime = !this.isDropDownEndTime;
11      this.isDropDownStartTime = false;
12    }
13  }
```

Quelltext 4.19: create-appointment.ts

Der toggleDropdown()-Funktion werden werden als Übergabeparameter die Variablen dropDownMenu und hour übergeben. Wenn das Dropdown Menü, um die Start Zeit des Termins auszuwählen geöffnet wurde, dann wird die darin gewählte Uhrzeit in die Variable startTimeValue geschrieben und das DropDown Menü geschloßen. Während das Start Zeit DropDown Menü geöffnet ist, darf das DropDown Menü um das Ende festzulegen nicht gleichzeitig geöffnet sein, da sich diese sonst überschneiden würden, deshalb wurde 'isDropDropDownEndTime' auf false gesetzt (Quelltext 4.18, Zeile 1-7). Für das DropDown Menü, um das Ende des Termins festzulegen, gilt das gleiche, wie für das DropDown Menü, in dem die Startzeit festgelegt wird, nur umgekehrt(Quelltext 4.18, Zeile 8-11).

Im HTML sehen die DropDown Menüs folgendermaßen aus.

```
1         <div id="startDropDown">
2             <div id="startDropDown-pos"></div>
3             <button id="startDropDown-input"
                  (click)="toggleDropDown('dropDownStartTime',
                  selectedHour)"
4                  class="dropDownMenu1">{{ startTimeValue ?
                      startTimeValue : 'Terminbeginn festlegen'
                      }}</button>
5             <div id="dropDownStartTime"
                  [ngClass]="{'dropDown-content1': true, 'show':
                  isDropDownStartTime}">
6                  <a *ngFor="let hour of hours"
                      (click)="toggleDropDown('dropDownStartTime',
                      hour)" class="hour-option">{{
7                      hour }}</a>
8             </div>
9         </div>
```

Quelltext 4.20: create-appointment.ts

Hierfür wurde viel mit css gearbeitet. Durch die Angular (click)-Funnktion wird die Funktion toggleDropDown() aus dem TypeScript aufgerufen, diese wird ausgeführt, wenn man auf den Button klickt(Quelltext 4.19, Zeile 3.1-3.2). Auf dem Button wird 'Terminbeginn festlegen' angezeigt, solange noch keine Uhrzeit gewählt wurde. Wurde eine Uhrzeit ausgewählt, dann wird die gewählte Uhrzeit auf dem Button angezeigt(Quelltext 4.19, Zeile 4-4.2). Durch die als Übergabeparameter festgelegte Variable dropDownStartTime wird der Container, der die Uhrzeiten enthält angezeigt, durch dropDown-content1': true und 'show': isDropDownStartTime(Quelltext 4.19, Zeile 5-8). Durch die ngFor werden die Stunden aus dem Array hours durchlaufen(Quelltext 4.19, Zeile 6). Die Stunden des Arrays werden dann als Button anzeigt. Wird auf einen Button geklickt des DropDown Menüs geklickt, dann wird durch die Angular (click)-Funktion nochmals die toggleDropDown()-Funktion mit den Übergabeparametern dropDownStartTime und hour aufgerufen, hour ist in dem Fall die aus dem DropDown ausgewählte Stunde, um das DropDown Menü wieder zu schließen(Quelltext 4.19, Zeile 6.1-6.2).

Dann kann mithilfe einer Checkbox festgelegt werden, ob es sich um einen öffentlichen oder privaten Termin handelt. Hierbei war ursprünglich geplant, dass wenn es sich um einen öffentlichen, dass dann alle angemeldeten Benutzer angezeigt werden und das daran dann ausgewählt werden kann, wer zu dem Termin hinzugefügt werden soll. Das Problem bei der Sache ist, dass es leider nur bedingt umsetzbar ist. Es gibt 2 verschiedene Möglichkeiten, um alle User für einen einzelnen User zugänglich zu machen.

Die 1. Möglichkeit wäre das alle User über Admin Rechte verfügen, da man sommit im Frontend Zugriff auf alle Benutzer haben kann. Wenn allerdings jeder einzelne Benutzer über Admin Rechte verfügt, dann heißt dass auch, das dieser Benutzer Zugriff auf die Administrationsoberfläche von Keycloak hat. Über der Administrationsoberfläche von Keycloak, kann das ganze Backend von Keycloak verwaltet werden, unter anderem können Benutzer gelöscht, Passwörter geändert werden etc. Deshalb kam diese Möglichkeit schon mal nicht in Frage.

Die 2.Möglichkeit wäre ohne einen Admin Account über eine REST Schnittstelle, dies wurde auch mitihlfe von Postman getestet und war soweit umsetzbar. Das Problem ist, dass über diesen Weg in der Administrationsoberfläche von Keycloak unter realm-management/Role Mappings für jeden einzelnen Benutzer die view-user Rolle zugewiesen werden muss. Dies würde wiederum bedeuten, dass man als Admin die Rolle view-users jedem einzelnen Benutzer übergeben muss, was bis auf etwas Zeitaufwand kein Problem ist. Allerdings besteht das Hauptproblem darin, dass der Admin dies nicht nur einmalig machen muss, sondern jedes mal neu sobald sich ein Benutzer neu registriert. Ein neu registrierter Nutzer landet in einer Liste ohne die 'spezielle Rolle' view-users zu Beginn. Für den Neuregistrierten wäre dann auch bis ihm die Rolle von einem Admin zugewiesen wird die Teamkalender App gar nicht im vollen Umfang nutzbar, da ihm für ursprüngliche geplante Team Termine, jetzt öffentliche Termine keine weiteren Benutzer angezeigt werden und somit kein Team Termin erstellt werden kann. Daher ist diese Möglichkeit zwar umsetzbar aber ohne Admin, der täglich prüft, ob sich ein neuer Benutzer registriert hat und die Rolle view-users für jeden neu Registrierten Benutzer zuweist nicht möglich [10].

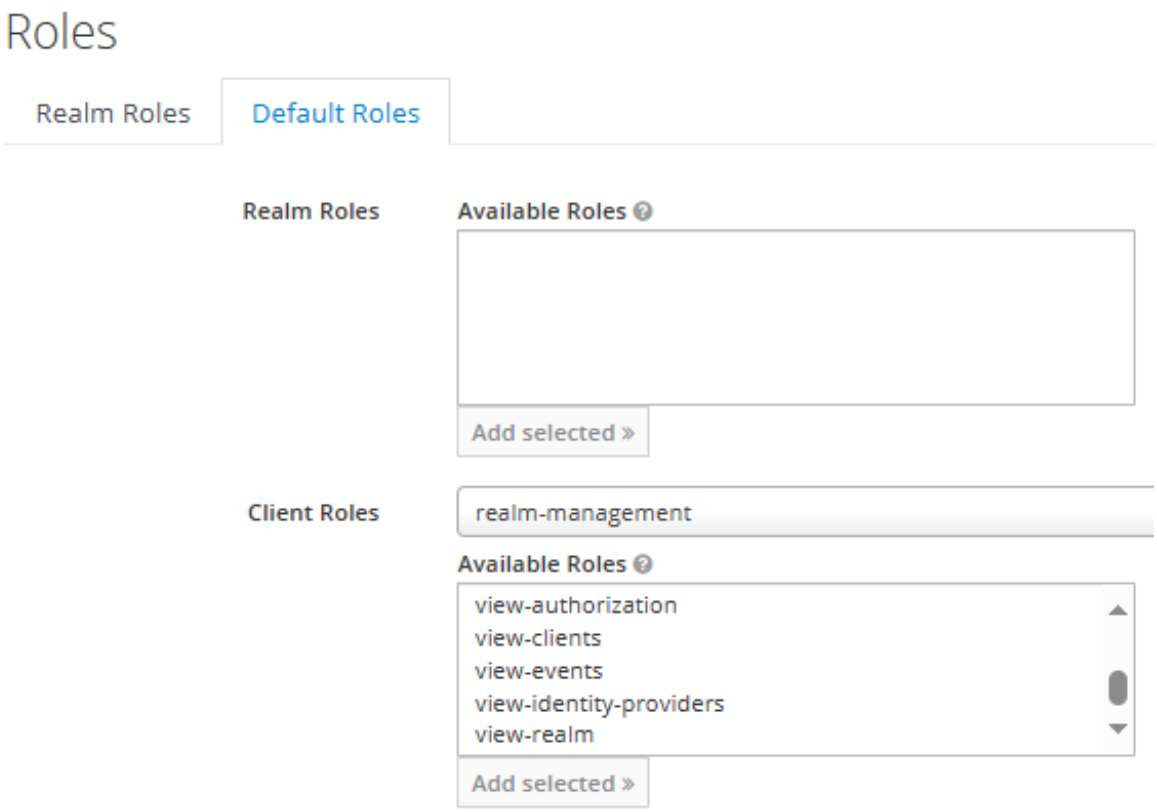


Abbildung 4.6: Benutzer Rollen

5 Test

Zum Testen des Backends wurde Postman genutzt. Da durch Postman gezielt die Backend Requests getestet werden konnten. Get Request:

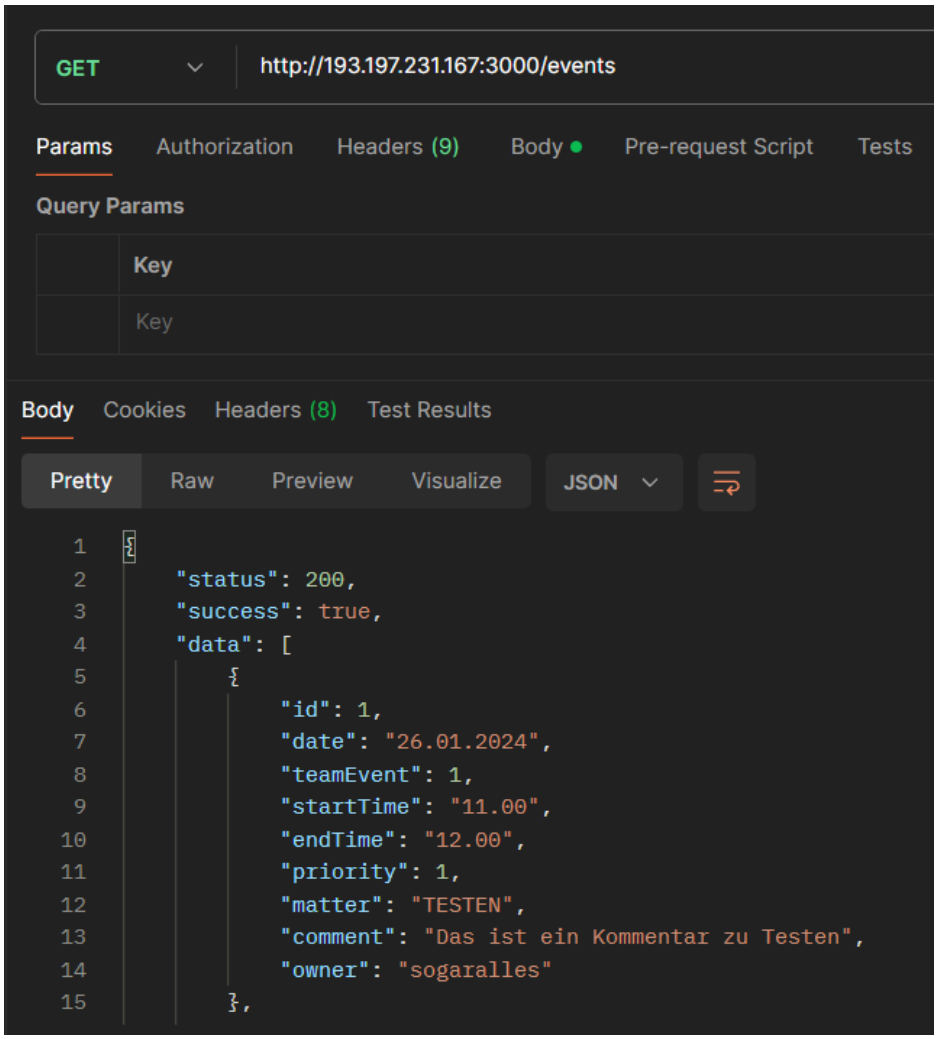


Abbildung 5.1: Postman GET Request

Post Request:

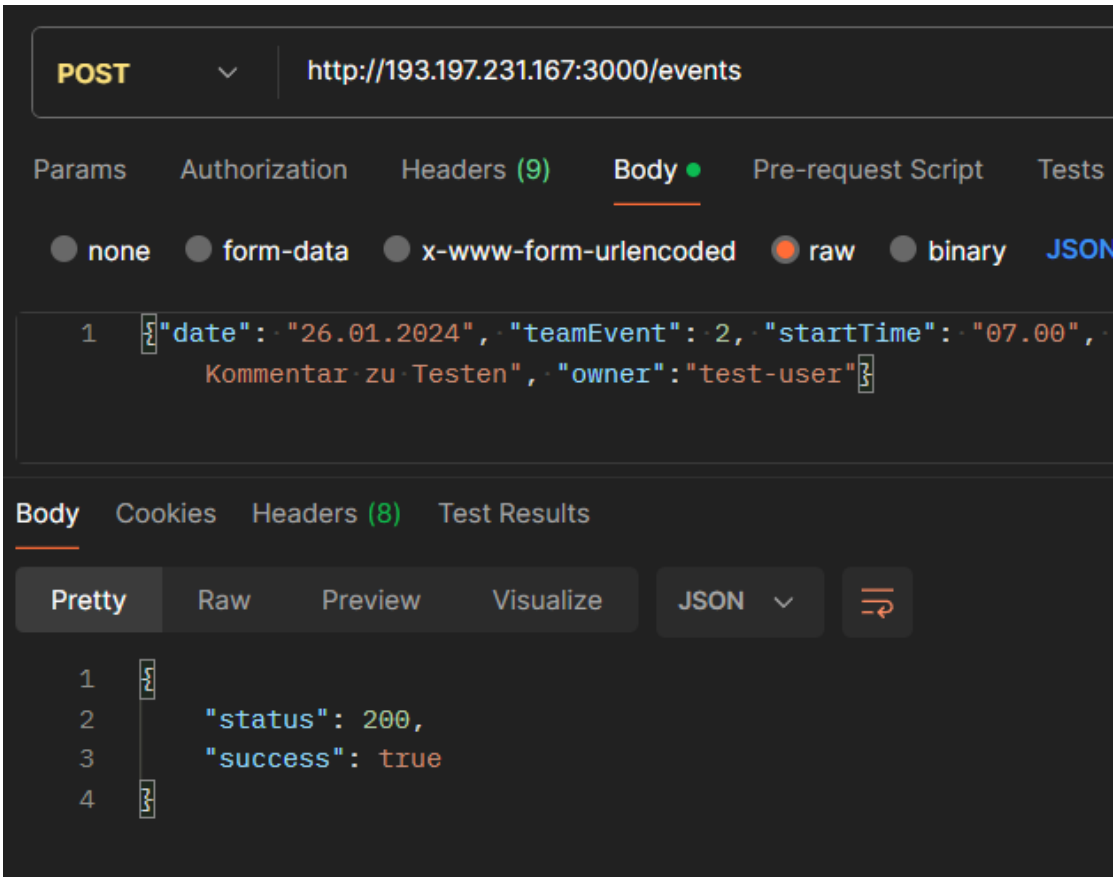


Abbildung 5.2: Postman POST Request

Dabei wurde folgendes als Body übergeben:
'date': '26.01.2024', 'teamEvent': 2, 'startTime': '07.00', 'endTime':'18.00' ,'priority': 1, 'matter': 'TESTEN', 'comment': 'Das ist ein Kommentar zu Testen', 'owner':'test-user'

Das wären die Variablen und deren Values. Die Values würde man im create-Appointment eingeben.

6 Inbetriebnahme

Damit die Teamkalender Web-App auf mehreren Endgeräten gleichzeitig zur Verfügung steht, wird ein virtueller Server, eine Virtuelle Maschine oder ein Raspberry Pi benötigt, ich habe bspw. BWCloud auf dem ein virtueller Server zur Vefügung gestellt wurde.

Um auf BWCloud einen virtuellen Server aufzusetzen muss folgendermaßen vorgegangen werden:

Zuerst muss unter Schlüsselpaare ein Schlüsselpaar erstellt werden.

Dazu muss ein Schlüsselpaarname eingegeben und der Schlüsseltp 'SSH Schlüssel' gewählt werden.

Zu dem darin generierten öffentlichen key muss ein privater key generiert und als Datei abgespeichert werden. Das ist mithilfe von Puttygen möglich.

-
- Unter Instanzen Instanz -> Instanz starten wählen
- Unter Details Instanzname eingeben, Anzahl = 1
- Unter Quelle Bootquelle = Abbild und unten Ubuntu 20.04 auswählen
- Unter Variante m1.tiny nutzen
- Unter Netzwerke Voreinstellungen lassen
- Unter Sicherheitsgruppe Voreinstellungen lassen
- Unter Schlüsselpaar erstelltes Schlüsselpaar auswählen
- Instanz starten auswählen

Dann kann im Putty die IP des virtuellen Servers eingegeben werden, zusammen mit dem Port 22. Zusätzlich muss unter Connection/SSH/Auth/Credentials der generierte private Key als Datei ausgewählt werden. Mit open wird dann der virtuelle Server geöffnet. Um sich einzuloggen wird dann nur der Benutzername 'ubuntu' benötigt.

Auf dem virtuellen Server kann dann das github Projekt gepullt werden.

Die Server IP muss vorab an ein paar Stellen im Code verändert werden. In der keycloak.init()-Funktion von der app.module.ts

```
1 url: 'http://193.197.231.167:8080/auth', //bwcloud IP can be
     changed
2         realm: 'teamkalender-realm',
3         clientId: 'teamkalender_client'
4       },
5       initOptions: {
6         redirectUri: 'http://193.197.231.167:4200',//bwcloud IP
             can be changed
```

Quelltext 6.1: app.module.ts

Und für die POST und GET Requests in der calender.component.ts, create-appointment.component.ts, daily-appointment.component.ts und in der home.component.ts.

Für die NodeJS API DB:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
nvm install 16.14.2

npm install express

Docker installieren:
Sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755-d /etc/apt/keyrings
sudo curl -fsSL
https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg –print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu $(. /etc/os-release echo"$VERSION_CODENAME")
stable
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
VERSION_STRING=5:24.0.7-1 ubuntu.20.04 focal
sudo apt-get install docker-ce=$VERSION_STRING docker-ce-cli=$VERSION_STRING
containerd.io docker-buildx-plugin docker-compose-plugin

Agular CLI installieren: npm install -g @angular/cli@14

Dann muss das Docker-image für das Frontend gebaut werden:
Frontend Docker Image lokal bauen und auf Server schieben, falls Performance abhängig docker build auf dem Server zu lange dauert(ist für bwcloud der Fall):

docker build -t <imageName> . docker save -o <tarName>.tar <imageName>

mit WinScp auf Server von bwcloud schieben auf bwcloud server:

docker load -i <tarName>.tar
docker run -d -p 4200:80 <imageName>

Keycloak starten: cd Teamkalender/keycloak/ sudo docker-compose up -d docker ps docker exec -it contaierID bash cd /opt/jboss/keycloak/bin ./kcadm.sh config credentials –server http://<ServerIP>:8080/auth –realm master –user admin1
Passwort: admin1

./kcadm.sh update realms/master -s sslRequired=NONE

Anschließend muss man sich auf der Administrationsoberfläche unter '<ServerIP>:8080' angemeldet werden: Benutzer: admin1 Passwort: admin1

Dann muss über Select realm, dass im Ordner keycloakRealm liegende Realm importiert werden. Wichtig dabei ist der Name: 'teamkalender-realm' und das enabled: 'on' ist. Dann den create Button klicken.

Wichtig ist: Name: 'teamkalender-realm' enabled: on Und unter Realm Settings -> Login -> User registration: on

Da sich die bisherigen User in einer Datenbank befinden sind diese nicht in der JSON vom realm gespeichert.

Auch hierfür muss dann über den virtuellen Server ssl deaktiviert werden:
./kcadm.sh update realms/teamkalender-realm -s sslRequired=NONE

Dann kann das Frontend über 'http://<ServerIP>:4200' aufgerufen werden.

Mit 'exit' kann der laufende docker-container dann wieder verlassen werden.

Backend kann mit folgenden Befehlen im Hintergrund gestartet werden: cd .. cd backend/ node app.js &

Dann kann das Frontend über '<ServerIP>:4200' aufgerufen werden und sich Registriert werden, um die Teamkalender Web-App nutzen zu können.

7 Zusammenfassung und Ausblick

7.1 Erreichte Ergebnisse

Das Ziel war eine Teamkalender Web-App, die für Zuhause oder auch in kleinen Firmen nutzbar ist. Sie sollte auf mehreren Engeräten gleichzeitigt nutzbar sein. Termine sollten zwischen privaten und öffentlichen unterschiedenen werden können, sodass es Termine gibt die für alle relevant sind oder auch Termine die geheim gehalten werden soll und nur für sich selbst sichtbar sind. Dazu sollten für öffentliche Termine andere Personen ausgewählt und eingeladen werden können, was leider nur bedingt umsetzbar war. CSS Decorators, dass die Web-App auch auf dem Handy nutzbar ist sind leider zeitlich nicht mehr Zustande gekommen.

7.2 Ausblick

Da es sich um eine Web-App handelt, die immer wieder weiterentwickelt werden kann, können auch in Zukunft weitere Funktionen hinzugefügt werden. Man muss nur auf die Ideen kommen, die mit der Zeit kommen, wenn man die Teamkalender Web-App öfter nutzt. Man könnte Termine bspw. noch eine Funktion um Termine bearbeiten zu können noch einbauen.

7.2.1 Erweiterbarkeit der Ergebnisse

Erweiterbar ist die Teamkalender Web-App an mehreren Stellen, da man auch Funktionalitäten von anderen Web-Apps mit einbauen könnte wie bspw. einen Chat hinzufügt, mit dem man mit anderen Chaten kann. Dadurch würde die App nicht mehr genau einem Thema zugeordnet sein. Durch Docker können auch unterschiedliche Backends eingebunden werden, sodass anderer Inhalt noch mit eingebunden werden kann.

7.2.2 Übertragbarkeit der Ergebnisse

Da die Web-App mit Docker läuft kann sie mit unterschiedlichen IPs und unterschiedlichen Ports genutzt werden, daher gibt es hier verschiedene Möglichkeiten die Web-App zu nutzen.

Literatur

[1] Axure RP. *Creating and Using Components*. 2024. URL: https://docs.axure.com/axure-rp/reference/creating-and-using-components/ (besucht am 26. 02. 2024).

[2] Axure RP. *The Axure RP Environment*. 2024. URL: https://docs.axure.com/axure-rp/reference/environment/ (besucht am 26. 02. 2024).

[3] Axure RP. *Viewing and Sharing Your Prototypes*. 2024. URL: https://docs.axure.com/axure-rp/reference/viewing-sharing-prototypes/#:~:text=You'll%20view%20and%20interact,as%20%22the%20HTML%20output. (besucht am 26. 02. 2024).

[4] bwCloud. *Erste Schritte - Wie komme ich in die bwCloud?* 2023. URL: https://www.bw-cloud.org/de/erste_schritte#:~:text=Die%20bwCloud%20ist%20eine%20%22Infrastructure,erzeugt%2C%20gestartet%20und%20betrieben%20werden. (besucht am 26. 02. 2024).

[5] Keycloak. *Keycloak features and concepts*. 2024. URL: https://www.keycloak.org/docs/latest/server_admin/index.html#keycloak-features-and-concepts (besucht am 26. 02. 2024).

[6] Mauricio Vigolo und Jon Koops. *Keycloak Angular*. 2024. URL: https://www.npmjs.com/package/keycloak-angular?activeTab=readme (besucht am 26. 02. 2024).

[7] Netguru. *What Is Node.js? Complex Guide for 2023*. 2024. URL: https://www.netguru.com/glossary/node-js#:~:text=It%20is%20used%20for%20server,a%20JS%20engine%2C%20and%20node (besucht am 26. 02. 2024).

[8] Postman. *What is Postman?* 2024. URL: https://www.postman.com/product/what-is-postman/ (besucht am 26. 02. 2024).

[9] Sebastian Wette. *Einführung in Angular*. 2021. URL: https://deduce.de/einfuhrung-in-angular/ (besucht am 26. 02. 2024).

[10] Stackoverflow. *How to get Keycloak users via REST without admin account*. 2017. URL: https://stackoverflow.com/questions/46470477/how-to-get-keycloak-users-via-rest-without-admin-account/46558530#46558530 (besucht am 20. 01. 2024).