Teilprojekt: Entwickeln eines Moduls zur Verwaltung des digitalen Klassenbuchs für das Gesamtprojekt „Academy-RP“, eines Umschulungsunternehmens

IHK Heilbronn
Winterprüfung 2022

Auszubildender:
Tobias Wehrle

Ausbildungsbetrieb:
Lutz und Grub GmbH

[redacted]

LUTZ GRUB ACADEMY

Inhalte

Projektbeschreibung 1
Projektumfeld 1
Ist-Zustand 1
Soll-Konzept 1
Anforderungen 2
Listenansicht 2
Neuer Eintrag 2
Detailansicht 2
Schnittstellen 2
Projektplanung 3
Phasenplan 3
Abweichungen vom Antrag 3
Entwurf 3
Datenmodell 3
Angular 3
Angular Material 4
Microsoft Authentication Library (MSAL) 4
Durchführung 5
Versionierung 5
Authentifizierung und Authorisierung 5
Routing und Zugriffsbeschränkung 6
Komponenten 8
Listenansicht 8
Neuer Eintrag Ansicht 9
Detailansicht 10
Register-Entry Service 12
Observables und rxjs 12
Quellenangaben 14
Anhänge 14
Zeitplanung 14
Interfaces 15
Formelemente für neuen Eintrag 16
Abbildungsverzeichnis 18

Projektbeschreibung

Projektumfeld

Das Projekt wurde im Rahmen meiner Umschulung zum Fachinformatiker bei der Lutz & Grub GmbH geplant und umgesetzt.

Die Lutz & Grub Academy ist ein Umschulungsunternehmen mit mehreren Standorten in Deutschland. Sie ist Teil der Lutz & Grub GmbH, die 1996 gegründet wurde und bietet Fortbildungen und Umschulungen im Bereich IT an. Außerdem gibt es neben der Academy noch die Lutz & Grub IT Services, welche IT Lösungen und IT Security für Firmenkunden anbietet. Zu den Kunden zählen unter anderem IT Softwarehäuser, Versorgungsbetriebe und andere Dienstleister.

Der Leiter der internen IT beauftragte mich damit, eine neue Anwendung zur Verwaltung des digitalen Klassenbuchs zu erstellen.

Ist-Zustand

Aktuell werden Klassenbucheinträge bereits digital, über eine Webanwendung, geschrieben. Diese Einträge beinhalten unter anderem folgende Informationen:

- Inhalt der Unterrichtseinheit
- Datum des Einsatzes
- Teilnahmen und Fehlzeiten der Teilnehmer

Außerdem ist die Zugriffsbeschränkung für die Unterschiedlichen Rollen für Benutzer umständlich und soll von nun an automatisch durch eine Authentifizierung bei Azure AD passieren.
Die Anwendung ist mittlerweile einige Jahre alt und nicht mehr an die heutigen Anforderungen angepasst.

Soll-Konzept

Um die Verwaltung der Lutz & Grub GmbH zu vereinfachen und an neue gegebenheiten anzupassen, ist gerade ein Projekt in der Umsetzung, das „Academy RP“ (oder „Academy Ressource Planning“) System.

Da die komplette Verwaltung in Zukunft über meherer eigens dafür erstellten Webanwendungen erfolgen soll, welche auf die gleiche API zugreifen, ist Ziel dieses Projekts eine neue Anwendung für das Klassenbuch zu schreiben, die sich in dieses „Academy RP“ System integriert und das „alte“ digitale Klassenbuch ersetzt.
Die Software soll, wie die anderen Module auch, als Webanwendung in Angular umgesetzt werden. Durch den Fokus auf diesem Framework, wird es einfacher für andere Kollegen, welche in Zukunft an anderen Modulen arbeiten, sich zurecht zu finden und die Umsetzung der Projekte beschleunigen.

Ziel des Projekts ist eine Webanwendung, welche die gleichen Funktionalitäten wie das alte digitale Klassenbuch bietet, inklusive der anpassung der Datengrundlage, einer neuen

implementation von Authorisierung und Authentifizierung und der verwendung der Angular Material Library.

Anforderungen

Die Anwendung soll dem Benutzer folgende Ansichten für Benutzer bereit stellen:

- Eine Listenansicht von allen Einträgen eines Kurses
- Eine Ansicht für Trainer, um einen neuen Eintrag zu erstellen
- Eine Detailansicht, wo Nutzer mit entsprechenden Berechtigungen Einträge bearbeiten können

Die Anwendung soll außerdem über eine Authentifizierung und Authorisierung verfügen, welche über Azure AD erfolgt. Dort sind die Accounts für alle Mitarbeiter der Lutz & Grub GmbH, zusammen mit deren Berechtigungen, hinterlegt. Die Endpoints der API prüfen zwar Serverseitig die Berechtigung, zusätzlich sollen Nutzer aber nur auf Ansichten zugreifen können, für die sie Berechtigt sind.

Listenansicht

In dieser Ansicht sollen alle Einträge eines Kurses angezeigt werden. Die Auswahl des Kurses erfolgt über zwei Select-Elemente, eines für den Standort und das andere für den Kurs.

Neuer Eintrag

Diese Ansicht beinhaltet zwei Listboxen, um die Teilnehmenden Kurse auszuwählen und ein Formular, mit dem ein neuer Eintrag erstellt werden kann. Folgende Informationen müssen dafür vom Benutzer, neben den Kursen, noch angegeben werden:

- Die Bezeichnung des Einsatzes (z.B. „JavaScript Grundlagen“)
- Die Bezeichnung des Moduls (z.B. „Primitive Datentypen“)
- Den Inhalt des Unterrichts

Nach einem Klick auf den Übernehmen Button, werden die Informationen aus dem Formular genommen, und in einem Datentransferobjekt an die API gesendet.

Detailansicht

In dieser Ansicht wird oben der Eintrag, mit allen Informationen wie vom Benutzer angegeben, angezeigt und darunter eine Liste aus Expansion Panels, eines für jeden Kurs. Jedes dieser Panels kann aufgekappt werden, um eine weitere Liste aus Expansion Panels anzuzeigen, eines für jeden Teilnehmer. Wenn diese wiederum geöffnet werden, werden hier alle Fehlzeiten angezeigt, welche in diesem Eintrag vermerkt. Dazu gibt es natürlich die Option neue Fehlzeiten hinzuzufügen.

Darüber hinaus besteht die Möglichkeit einzelne Teilnehmer ganztägig auf Abwesend zu stellen. Dies ist für die spätere hinzukommende Auswertung der Abwesenheiten wichtig.

Schnittstellen

Als Datengrundlage dient eine bereits entwickelte Datenbank und einer API, welche in ASP.NET C# geschrieben wurde und an der aktuell auch noch gearbeitet wird.
Die Webanwendung für das digitale Klassenbuch kommuniziert ausschließlich mit der API und der Azure ID, nicht direkt mit der Datenbank. Dies stellt sicher, dass ein Nutzer wirklich

nur Daten verändern kann, für die er berechtigt ist, indem die API die Berechtigung des Benutzers nochmals überprüft.

Projektplanung

Phasenplan

Das Projekt wurde innerhalb von 80 Stunden durchgeführt. Eine genaue Zeitplanung findet sich im Anhang unter Zeitplanung.

Abweichungen vom Antrag

Aus dem Projektantrag geht hervor, dass die Programmierung des für die Webapp zu verwendeten API-Endpoints Teil des Projekts ist. In der Zeit zwischen dem Antrag und dem Beginn der Durchführung wurde allerdings der entsprechende Endpunkt bereits erstellt, dieser war allerdings nicht ganz fertig.
Da die Programmierung des API-Endpoints bereits von einem Kollegen übernommen wurde, und demnach nicht mehr Teil dieses Teilprojektes ist, wurde sich auf die Webapplikation konzentriert. Die für den Endpoint geschätzte Zeit wurde, innerhalb der Durchführung, auf die Programmierung des Services, der Komponenten und der im Projektantrag nicht betrachteten Implementation von Authentifizierung/Authorisierung umgelegt.

Entwurf

Da eine „alte“ Version der zu entwickelnden Anwendung bereits existiert, mit der die Nutzer vertraut sind, wurde sich schnell dafür entschieden, das Design weitesgehend unverändert zu lassen. Dies lässt sich, da es sich um ein internes Projekt handelt, auch später noch anpassen, was durch bessere Dokumentation des Codes und moderne Entwicklungsansätze einfacher werden soll, als bei der bereits vorhandenen Lösung.

Datenmodell

Weil zur Datenabfrage ausschließlich die API dient, wurden bereits Endpoints erstellt, die die notwendigen Daten in Form von definierten DTOs (Daten-Transfer-Objekt) übertragen, welche für die Ansichten in der Webanwendung benötigt werden. Deswegen wurde, um in der Webanwendung sicher zu stellen, dass alle Daten wie erwartet vorhanden sind, Interfaces erstellt, welche den DTOs der API entsprechen. Ein Auszug einiger Interfaces findet sich in den Anhängen.

Angular

Angular ist, laut angular.io, der offiziellen Webseite:

- Ein Komponentenbasiertes Framework für skalierbare Webanwendungen.
- Eine Sammlung von Softwarebibliotheken, welche bei der Implementierung essentieller Features helfen (z.B. Routing, Formulare und mehr)
- Ein komplettes Packet inklusive Möglichkeiten zum Entwickeln, Building, Testen und Updaten der Codebase

Der Kern von Angular sind sogenannte Komponenten. Eine Komponente ist ein UI Element, inklusive des Notwendigen HTML, CSS und selbst funktionalität über JavaScript. Um es einfach zu Beschreiben: Man kann damit seine eigenen, selbstständigen UI Komponenten erstellen, inklusive notwendiger Schnittstellen, um diese wieder verwenden zu können.

Angular wird außerdem häufig als Single-Page-Application Framework bezeichnet. Das heißt essentiell, dass die Komplette Anwendung auf einer HTML Seite angezeigt wird, oder mit anderen Worten: Man wechselt nicht auf andere Seiten, sondern die aufgerufene Komponente wird geladen, basierend auf der Route in der URL.

Wie die Struktur und die Hilfsmittel von Angular in diesem Projekt Anwendung finden, wird in der Durchführung genauer erleutert.

Angular Material

Um das Design einheitlich zu halten und die Entwicklung der Module des „Academy RP“ Gesamtprojektes zu beschleunigen, wurde sich schon früh auf die „Angular Material“ Library geeinigt.
Angular Material ist eine Komponentenbibliothek für Benutzeroberflächen in Angular. Sie entspricht dem „Material Design“, welches von Google entwickelt wurde.
Es vereinfacht die Entwicklung, da es fertige UI Elemente beinhaltet, die mit wenig Aufwand auch an die jeweiligen Anforderungen angepasst werden können. Dabei bietet es allerdings mehr als vorgestylte HTML Elemente sondern auch komplexere Funktionen, wie z.B. Expansion Panels.

Abbildung 1 Beispiel Angular Material Expansion Panel

Informationen zu allen von Angular Material angebotenen Komponenten finden sich unter https://material.angular.io/components/.

Microsoft Authentication Library (MSAL)

Die Microsoft Authentication Library, oder kurz MSAL, ist ein npm package, mit welchem sich eine Anmeldung, in einer NodeJS Anwendung, einfach und unkompliziert implementieren lässt. Dadurch ist es möglich, den Nutzer über ein Popup oder einer Weiterleitung auf die Anmeldeseite von Microsoft weiter zu leiten, wo sich der Benutzer mit seinem Microsoft Azure AD Account anmelden kann. Dadurch wird ein Token erstellt, welches im LocalStorage des Webbrowsers abgespeichert wird. Damit hat die Applikation zugriff auf die Rollen, welche in Azure hinterlegt sind und auf welche Seiten der Nutzer zugriff hat.
Dies setzt allerdings voraus, dass bereits eine Rollen und Rechteverwaltung in Azure AD existiert. Da Lutz & Grub für Mitarbeiter und Teilnehmendene bereits Azure AD benutzt, muss für dieses Projekt lediglich die Anmeldung/Abmeldung über das MSAL Package implementiert werden, und die Routen dementsprechend abgesichert werden.
Eine Anleitung für die Implementierung findet sich unter learn.microsoft.com und die Implementation im Abschnitt Authentifizierung und Authorisierung.

Durchführung

Versionierung

Um sicher zu stellen, dass Änderungen nachvollziehbar bleiben und diese Rückgängig gemacht werden, wird für das Gesamtprojekt ein Git Repository genutzt, mit einem Github Repository für die Synchronisation unter den Entwicklern. Das digitale Klassenbuch wurde in diese Versionsverwaltung mit aufgenommen.

Authentifizierung und Authorisierung

Die erste Herausforderung des Projekts war die Authentifizierung und Authorisierung über Azure AD. Wie unter Entwurf dargestellt war klar, dass wir damit arbeiten und die API hat diese Bibliothek bereits implementiert, allerdings habe ich selbst vorher nie mit Azure AD gearbeitet. Mit Hilfe der Kollegen, welche die API Authentifizierung implementiert haben, haben wir es geschaft das MSAL Package mit unserem Active Directory zu verbinden.

```
ngOnInit(): void {
  // Apparently it's not recommended to perform multiple msal actions at once, so this check makes sure
  // that msal is not currently doing anything
  this.msalBroadcastService.inProgress$
    .pipe(
      // Monitor the msalService and only act, when it is not currently doing anything
      filter((status: InteractionStatus) => status === InteractionStatus.None),
      takeUntil(this.unsubscribe)
    )
    .subscribe(() => {
      this.setAuthenticationStatus();
    })
  this.msalBroadcastService.msalSubject$
    .pipe(
      filter((message: EventMessage) => message.eventType === EventType.LOGIN_SUCCESS),
      takeUntil(this.unsubscribe)
    ).subscribe((message: EventMessage) => {
      const authResult = message.payload as AuthenticationResult;
      this.msalService.instance.setActiveAccount(authResult.account);
    })
}
```

```
login() {
  this.msalService.loginRedirect();
}

logout() {
  this.msalService.logoutRedirect();
}
```

Abbildung 2 Implementation von MSAL in app.component.ts

Sobald die Seite aufgerufen wird, wird bereits der Login Prozess gestartet. Falls seit der Nutzer mit dem Browser bereits mit einem Berechtigten Account angemeldet ist, dann wird er automatisch auch dem digitalen Klassenbuch angemeldet, ohne dass zusätzliche Interaktion des Nutzers notwendig ist.
Falls das MSAL Package keinen gespeicherten Login oder mehrere Accounts findet, wird der Nutzer auf die Login Seite von Microsoft weitergeleitet und nach einem erfolgreichen Login wieder auf die Seite des Klassenbuchs zurück geleitet.

Zusätzlich wurde aber noch ein Login/Logout Button mit eingebaut, welcher es dem Nutzer ermöglicht sich Abzumelden, oder den Login manuell einzuleiten.

Wie die Beschränkung von Benutzern auf die für sie vorgesehene Ansichten erfolgt, wird im folgenden Teil genauer erleutert.

Routing und Zugriffsbeschränkung

Wie im Abschnitt Entwurf angesprochen, erfolgt die Navigation innerhalb der App über das Angular Routing Modell. Das Prinzip ist recht einfach erklärt:

```
<mat-toolbar color="primary">
  <mat-toolbar-row>
    <span routerLink="/">Klassenbuch</span>
    <span class="spacer"></span>
    <button routerLink="/entries" mat-flat-button class="nav-button">Einträge anzeigen</button>
    <button mat-flat-button class="nav-button">Eintrag erstellen</button>
    <span>
      <a>{{activeUser}}</a>
      <button *ngIf="!isAuthenticated" (click)="login()" mat-flat-button color="accent" class="nav-button login">Login</
      button>
      <button *ngIf="isAuthenticated" (click)="logout()" mat-flat-button color="warn" class="nav-button logout">Logout</
      button>
    </span>
  </mat-toolbar-row>
</mat-toolbar>
<div>
  <h1>Test API-Call</h1>
  <button (click)="callAPI()">Call ASP-API</button>
  <button (click)="callMSAPI()">Call MSGraph-API</button>
  <div>
    <p>{{apiTestData | async | json}}</p>
  </div>
</div>

<router-outlet></router-outlet>
```

Abbildung 3 Router-Outlet in app.component.html

In dieser Abbildung ist die app.component.html dargestellt, welche die HTML Seite die der Nutzer sieht wiederspiegelt. Allerdings ist hier nur die Navigationsleiste zu sehen und ein Element, mit der Bezeichnung „router-outlet“. Das ist das Element, in dem die Komponente geladen wird, welche wir mit unserem Router ansteuern.

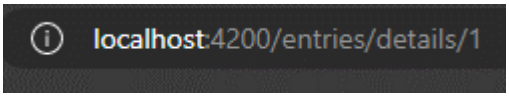

Abbildung 4 Beispiel einer URL Route

Wie im Abbild zu erkennen ist, haben wir eine URL wie „/entries/1“ können wir daraus lesen, dass der Aufrufer die Seite für dem Eintrag mit der ID 1 aufrufen möchte. Also zeigen wir dem Aufrufer der Seite die Benutzer Details Komponente, und übergeben diese ID (in diesem Beispiel „3“) an die Details Komponente.

Die Routen werden in der „app-routing.module.ts“ definiert, wie in folgender Abbildung beispielhaft dargestellt.

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  {
    path: 'entries',
    canActivate: [MsalGuard],
    // this data attribute is for limiting the route to users with the given roles.
    // only works when route is guarded by msal.
    data: {
      roles: [
        Role[Role.Trainer],
        Role[Role.Coach],
      ]
    },
    children: [
      {
        path: '',
        component: ListEntriesComponent,
      },
      {
        path: 'new',
        component: NewEntryComponent
      },
      {
        path: 'details/:id',
        component: EntryDetailsComponent
      }
    ]
  },
```

Abbildung 5 Angular Router Code-Snippet

Das Routing Modell von Angular ist hirarchisch aufgebaut, das heißt Routen können untergeordnete Routen haben. In diesem Beispiel haben wir die „entries“ Route. Wenn der Benutzer auf „/entries“ navigiert, oder durch das klicken von Links von der Anwendung Navigiert wird, dann wird die darin definierte Komponente geladen. In diesem Fall haben wir für die „/entries“ Route aber 3 untergeordnete Routen, welche im „children“ Array deklariert sind.

In unserem Beispiel wollen wir auf die Entry Details, also die Detailansicht für einen Eintrag navigieren, welche in der Abbildung die letzte Unterroute darstellt. Die Übergabe der Information über die ID des Eintrags welcher angezeigt werden soll entspricht hier dem Teil „details/:id“ wobei der Doppelpunkt anzeigt, dass es sich hier um einen variablen Teil der Route handelt.

Im Übergeordneten Teil der Routendefinition findet sich zusätzlich noch ein „data“ Objekt mit einem Array an Rollen. Diese werden von der MSAL Guard verwendet und stellen eine Liste an Rollen dar, welche berechtigt sind auf diese Route zugreifen zu können.
Guards sind Implementationen des von Angular vorgegebenen Guard Interface. Sie haben eine Funktion, in der alle notwendigen Informationen über die Routen übergeben werden, welche aufgerufen wird, bevor der Angular Router die Komponente läd.
Diese Funktion gibt einen boolean Wert von True oder False zurück, je nachdem ob der Benutzer über eine der Rollen, welche in dem Array der erlaubten Rollen definiert sind, verfügt. Wird von dieser Guard Funktion der Wert „True“ zurückgegeben, dann ist der Benutzer berechtigt auf die untergeordneten Routen zu navigieren.

Komponenten

In diesem Abschnitt werden zur Veranschaulichung Abbildungen der jeweiligen Ansichten gezeigt. Es sollte dazu erwähnt werden, dass durch die Verzögerung der Fertigstellung der API und Zeitmangel als Datengrundlage nur limitierte Testdaten verwendet wurden und Tests mit den tatsächlich schon vorhanden Daten ausstehen. Dies wurde mit dem Projektleiter und dem Kunden abgesprochen.

Listenansicht

Da diese Ansicht die einfachste ist umzusetzen, wurde mit dieser Komponente begonnen. In dieser Ansicht sollen sich Trainer und Coaches eine Liste mit allen Einträgen eines Kurses anzeigen lassen können. Um das finden von Kursen für den Benutzer einfach und komfortabel zu halten, wurde sich eine Filterfunktion gewünscht.

Diese Komponente enthält zwei Select Elemente, wo im ersten Select der Standort und im zweiten der Kurs ausgewählt werden kann. Je nachdem welcher Standort im ersten Select ausgewählt wird, werden die dementsprechenden Kurse im zweiten Select geladen.
Beide Select Elemente haben ein Data-Binding mit jeweils einer Variablen, die dazu dient, den aktuellen ausgewählten Standort und Kurs zu registrieren, mit denen der Register-Entry-Service dann HTTP-Requests an die API senden und die relevanten Daten geladen werden. Mehr zur Funktion des Services und der HTTP-Requests ist im Abschnitt Register-Entry-Service zu finden.

Dem Kunden war für das Projekt wichtig, dass möglichst viele der UI Elemente sich, bei späteren Änderungen des Datenhintergrunds ohne Erweiterung des Programms anpassen können. Aus diesem Grund wurden die Standorte nicht fest mit einprogrammiert, sondern werden ebenfalls von der API abgefragt.

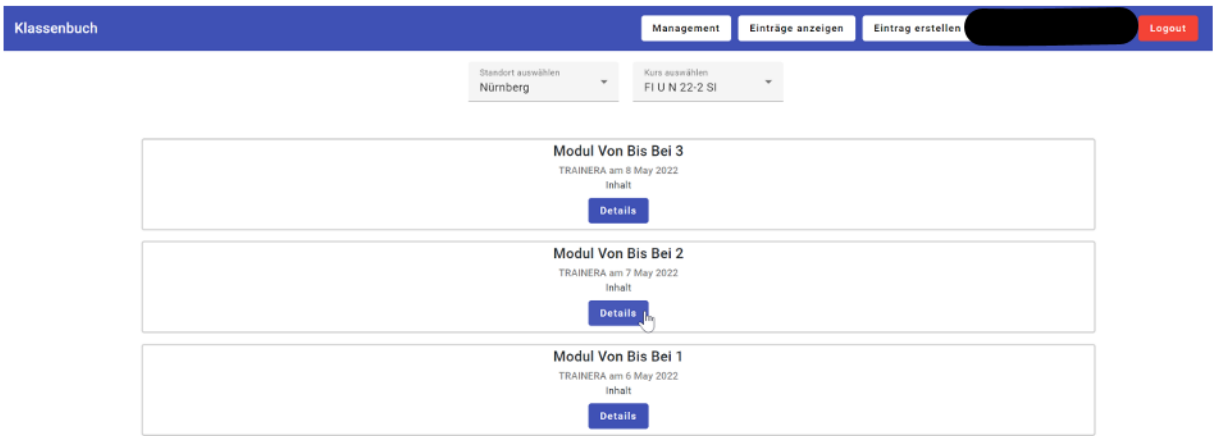

Abbildung 6 Listenansicht für alle Einträge

Neuer Eintrag Ansicht

Diese Ansicht ist bei weitem die wichtigste, da es sich hier um das Tagesgeschäft der Trainer*Innen und Coaches handelt. Für jeden Einsatz, welcher laut unserer Definition ein Unterrichtstag ist, ist ein Klassenbucheintrag erforderlich.

Um einen neuen Eintrag erstellen zu können, werden folgende Informationen benötigt:

- Eine Liste an Kursen, für die der Eintrag erstellt werden soll
- Eine Einsatzbezeichnung (z.B. „Arrays in JavaScript“)
- Eine Modulbezeichnung (z.B. „JavaScript Grundlagen“)
- Der Inhalt des Unterrichts (Aufgaben, Inhalte,...)

Die Einsatzbezeichnung, Modulbezeichnung und der Inhalt des Unterrichts waren einfach in einem Formular umzusetzen. Der kompliziertere Teil war die Auswahl der Kurse, denn diese sollten einfach auszuwählen und auch wieder entfernbar sein, da unter Umständen viele Kurse an einem Einsatz teilnehmen. Um dies für den Nutzer so einfach wie Möglich zu gestalten, kommen 2 Mat-List-Elemente zum Einsatz, zusammen mit Buttons für jeden Standort, um die Suche nach den Kursen einfacher zu gestalten.

Eine Mat-List, ist eine von Angular Material angebotene Komponente, die einen Container für eine unbeschränkte Anzahl von Elementen darstellt. Man kann sie am besten mit dem UL-Element, der „unordered list“, vergleichen.
Die Auswahl der Kurse funktioniert wie folgt:

- Der Benutzer nutzt die Buttons um sich die Kurse an einem Standort anzeigen zu lassen.
- Anschließend kann der Nutzer die Kurse in der ersten Liste anklicken, welches den Kurs aus der Liste entfernt und in die zweite Liste einfügt.
- Der Vorgang wird wiederholt, bis alle benötigten Kurse ausgewählt wurden.
- Falls der Benutzer einen falschen Kurs angeklickt hat, kann er diesen mit einem Klick auf den Kurs in der zweiten Liste wieder aus der Auswahl entfernen.

Das Sortieren der Kurse nach Standort funktioniert über eine Variable, welche die ID des aktuell ausgewählten Standortes darstellt. Diese Variable wird durch einen Funktionsaufruf bei einem Klick auf einen der Standort Buttons verändert. Für den Fall, dass der Benutzer sich alle Kurse in der Liste anzeigen lassen möchte, kann er dies ebenfalls durch einen Klick auf den „Alle“ Button, welche als StandortID „-1“ definiert ist. Die Funktion, welche die Kursliste nach Standort filtert, gibt die komplette Liste, also alle Kurse, zurück, wenn die StandortID „-1“ beträgt.

Sobald alle Informationen angegeben wurden, kann auf den „Übernehmen“ Button geklickt werden, um den Eintrag zu erstellen. Dazu wurde ein Interface erstellt, welches dem DTO aus dem Post-Request in der API entspricht. Ein neues Objekt wird erstellt, welches dem Interface entspricht, die Werte aus dem Formular werden in das Objekt gelesen und anschließend dem Register-Entry-Service für den Request übergeben. Falls der Vorgang erfolgreich abgeschlossen wird, wird der Benutzer automatisch auf die Detailansicht des Eintrags weitergeleitet.

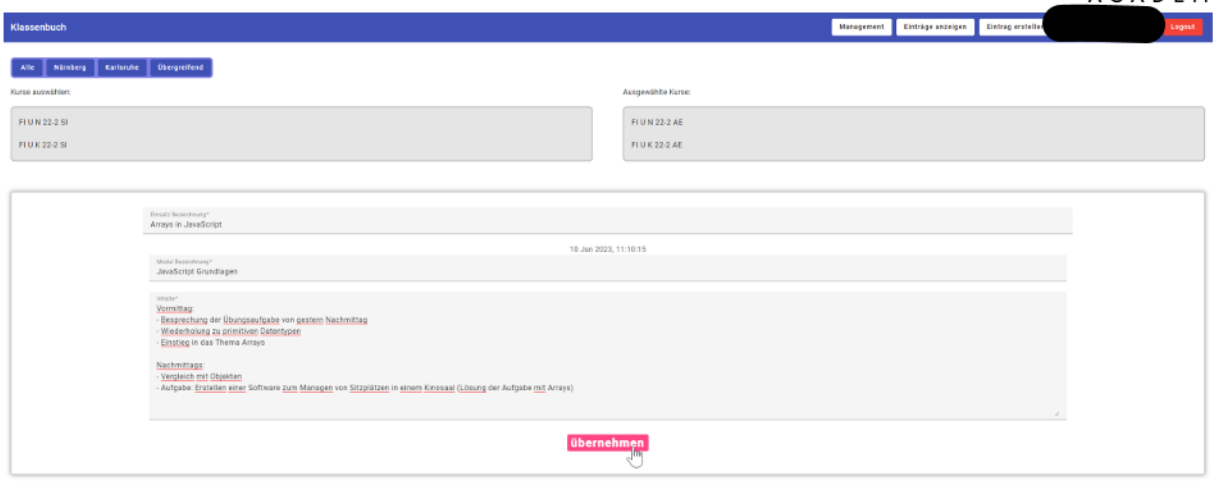

Abbildung 7 Ansicht für einen neuen Eintrag (alle Standorte, 2 Kurse selektiert)

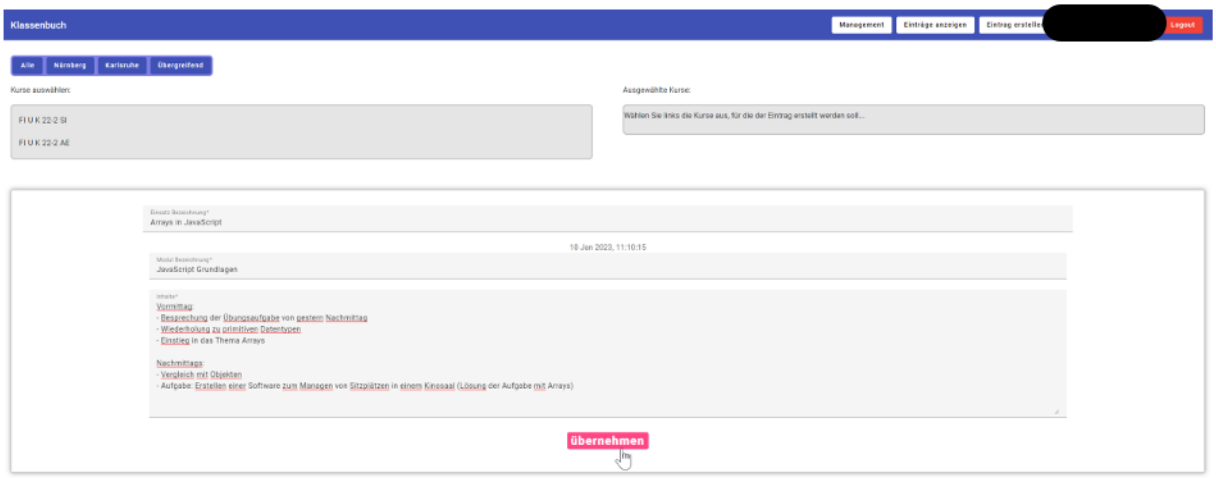

Abbildung 8 Ansicht für einen neuen Eintrag (filter nach Standort Karlsruhe, kein Kurs selektiert)

Detailansicht

In dieser Ansicht sollen Trainer und Coaches einen erstellten Eintrag bearbeiten können und die Anwesenheit der Teilnehmer erfassen können.

Die Seite ist in 2 Bereiche aufgeteilt. Oben findet sich der Klassenbucheintrag, für welche das Karten Element aus Angular Material gewählt wurde, wie in der Listenansicht auch. Besonderst ist hier, dass es einen „Edit“ Button gibt, mit die Textfelder in der Karte bearbeitbar werden. Nun kann der Benutzer die Einsatzbezeichnung, die Modulbezeichnung und den Inhalt bearbeiten. Wenn man mit den Änderungen zufrieden ist, kann man auf den Button „Übernehmen“, der nur in der Bearbeitungsansicht sichtbar ist, klicken um sie zu speichern.

Unter dem Klassenbucheintrag befindet sich eine Liste von Angular Material Expansion Panels, eines für jeden teilnehmenden Kurs. Im Header des Expansion Panels steht die Kursbezeichnung und ein Badge Element, ebenfalls aus Angular Material, welches die Anzahl der Teilnehmer auf einen Blick anzeigt. Sobald man auf einen Kurs klickt, öffnet sich das Expansion Panel und zeigt eine weitere Liste an Expansion Panels, welche die Teilnehmer darstellen. Im Header dieser Elemente stehen Vor-/Mittlerer- und Nachname, und wie bei den Kursen auch, ein Badge Element, welches die Anzahl der eingetragenen Fehlzeiten für diesen Einsatz anzeigt. Wenn dieses Expansion Panel nun geöffnet wird, zeigt es eine Tabelle mit allen Fehlzeiten und eine Zeile mit Input Elementen, um eine neue Fehlzeit einzutragen. Für eine neue Fehlzeit wird hier von Benutzer allerdings nur die Start und Endzeit benötigt. Die restlichen benötigten Daten für den HTTP-Request werden von der App von selbst ausgefüllt, aus den bekannten Daten.

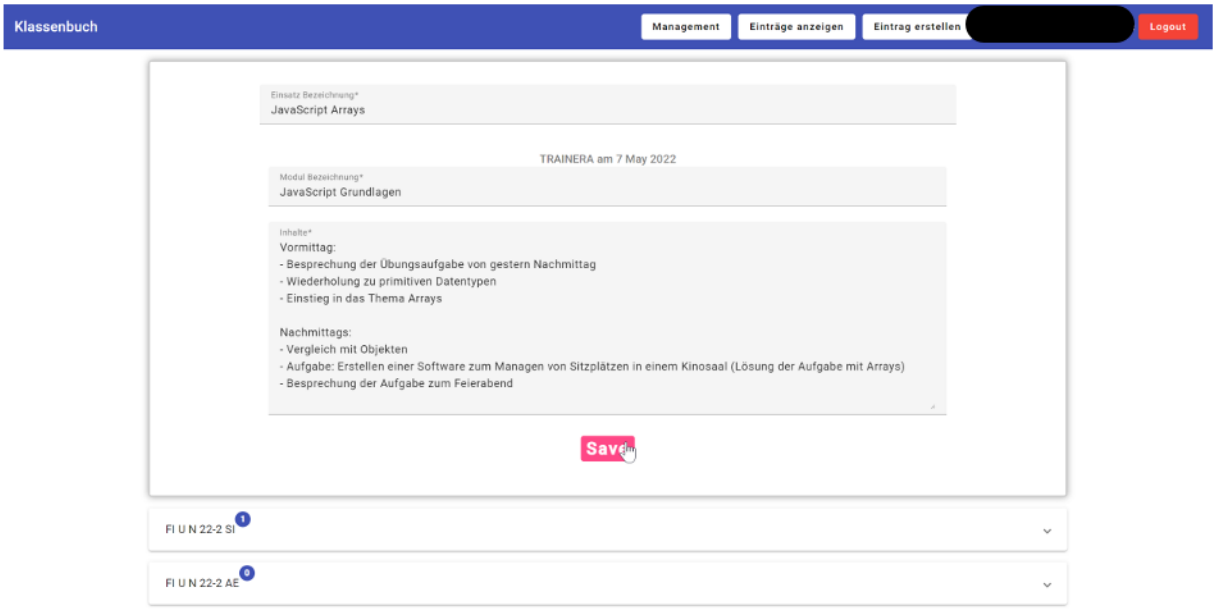

Abbildung 9 Detailansicht im Bearbeitungsmodus

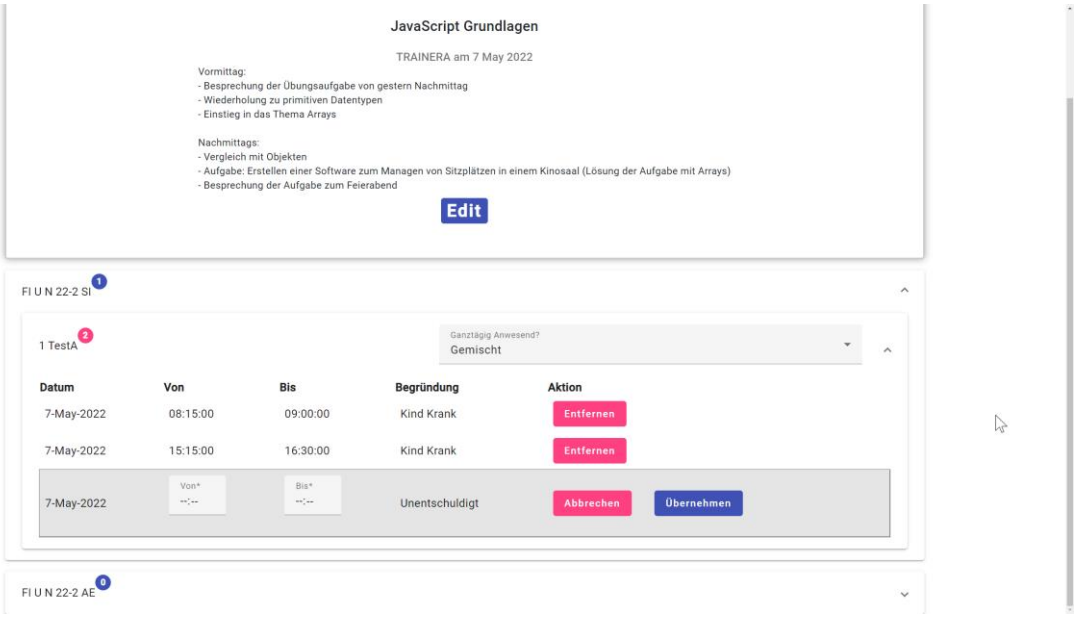

Abbildung 10 Detailansicht mit aufgeklapptem Kurs und Teilnehmer

Register-Entry Service

Um den Code besser wartbar zu gestalten, wurde ein Service erstellt, um die Programmlogik für die HTTP-Requests von denen, welche für die Ansicht der Daten zuständig ist auszulagern.
Der Service nutzt den Angular HTTP-Client für sämtliche Kommunikation mit der API. Dieser hat den Vorteil ein Observable zurückzugeben, welches über die für rxjs üblichen pipe und map operatoren einfachen Zugriff auf die Daten in der HTTP-Response befinden. Zusätzlich lassen sich HTTP-Fehler ganz einfach über den catchError Operator abfangen, welcher als Parameter eine Funktion erwartet, welche diese Fehler behandelt.

Observables und rxjs

Rxjs, oder auch „Reactive Extensions for JavaScript“, welches bei Angular bereits mitgeliefert wird und erweitert das Observer Design Pattern. Es erlaubt uns asynchrone Datenstreams zu erstellen und zu verarbeiten. Die Operatoren sind dabei die Transformationen, die an diesem Datenstream vorgenommen werden, befor die Daten an ihrem Ziel ankommen. Das Ziel ist dabei der Observer, oder derjenige, der das Observable „subscribed“. Sobald dies passiert, werden alle Daten aus dem Strom ausgegeben. Man kann sich das Observable am besten als Array über eine Zeitspanne vorstellen.

Die Operatoren, welche auf die Observables angewand werden können unter anderem transformieren, filtern oder aggregieren. Sie können miteinander verbunden werden um mehrere transformationen an einem Stream vorzunehmen.

Mehr Informationen zu den Reactive Extensions findet sich unter reactivex.io. Diese werden auch für andere Programmiersprachen bereitgestellt. (z.B. C#, C++, Python, Java,...)

```
// *****
// EntryList
// *****


/**
 * Gets called in entry/list-entries/list-entries.component.ts
 * @returns Observable<EntryListKursDetails[]>
 */
getEntryListAllKurse(): Observable<EntryListKursDetails[]> {
  return this.http.get<any>([redacted].pipe(
    map((data: any) => data.kurse),
    catchError(this.handleError),
  );
}

/**
 * Gets called in entry/list-entries/list-entries.component.ts
 * @returns Observable<EntryListStandort[]>
 */
getEntryListStandorte(): Observable<EntryListStandort[]> {
  return this.http.get<any>([redacted]).pipe(
    map((data: any) => data.standorte),
    catchError(this.handleError),
  );
}
```

Abbildung 11 Code-Snippet Register-Entry-Service Get-Requests

```
/**
 * Gets called in entry/list-entries/entry-new.component.ts
 * @returns Observable<HttpResponse>
 */
postEntryNewEintrag(entry: PostEntryNewKlassenbuchEintrag): Observable<any> {
  return this.http.post<any>([redacted], entry).pipe(
    catchError(this.handleError),
  );
}
```

Abbildung 12 Code-Snippet Register-Entry-Service Post-Requests

Quellenangaben

Anleitung zur Implementation von MSAL	Tutorial: Create an Angular app that uses the Microsoft identity platform for authentication using auth code flow - Microsoft Entra | Microsoft Learn
Angular Dokumentation	Angular - Introduction to the Angular Docs
Angular Material Dokumentation	Angular Material UI component library
MDN Web-Docs	Web technology for developers | MDN (mozilla.org)
Reactive Extensions Dokumentation	ReactiveX

Anhänge

Zeitplanung

Zeitplanung Grob

Projekt-Phasen	**Zeitaufwand in Stunden**
1. Ausgangssituation	5
2. Ressourcen- und Aufgabenplanung	2
3. Aus-/Durchführung	50
4.Projektergebnisse	5
5. Projektbegleitend	18

Tabelle 1 Zeitplanung Grob

Zeitplanung Detaliert

Projekt-Phasen	**Zeitaufwand in Stunden**
1. Ausgangssituation	**5**
1.1 Kundengespräch	1
1.2 Ist-Analyse	2
1.3 Lösungsmöglichkeiten/Sollkonzept	1
1.4 Projektziele definieren	1
2. Ressourcen- und Aufgabenplanung	**2**
2.1 Projektplanung (Personal, Kosten, Termine)	2
3. Aus-/Durchführung	**50**
3.1 Aufsetzen des Projekts / Abhängigkeiten einbinden	2
3.2 Implementierung der Authentifizierung und Authorisierung	8
3.3 Programmieren des Services für Kommunikation mit der API	14
3.4 Erstellen der Komponenten	3
3.5 Programmieren der Funktionalitäten der Komponenten	16
3.6 Styling-Regeln für Komponenten definieren	3
3.7 Testen der Web-App	4
4.Projektergebnisse	**5**
4.1 Abschlusstest	2

4.2 Soll-Ist Vergleich	1
4.3 Übergabe an den Kunden	2
5. Projektbegleitend	**18**
5.1 Troubleshooting	2
5.2 Kundendokumentation	3
5.3 Projektdokumentation	13
Zeitaufwand gesamt	**80**

Tabelle 2 Zeitplanung Detaliert

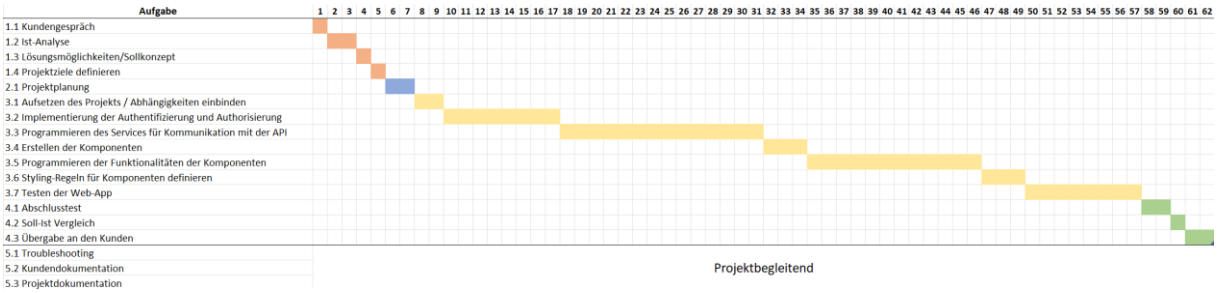

Abbildung 13 Zeitplanung als Gantt-Diagram (Zeitverlauf in Stunden)

Interfaces

```
export interface PostEntryNewKlassenbuchEintrag {
  einsatzBezeichnung?: string;
  modulBezeichnung?: string;
  klassenbuchEintraegeInhalt?: string;
  kurse?: number[];
}
```

Abbildung 14 Interface für neuen Klassenbucheintrag

```
export interface EntryListStandort {
  standorteId?: number;
  standorteName?: string;
}
```

Abbildung 15 Interface für Abfrage der Standorte

```typescript
export interface EntryDetailsKlassenbuchEintrag {
  klassenbuchEintraegeId?: number;
  klassenbuchEintraegeDatum?: string;
  einsaetzeId?: number;
  einsaetzeBezeichnung?: string;
  moduleBezeichnung?: string;
  klassenbuchEintraegeInhalt?: string;
  klassenbuchEintraegeTrainer?: EntryDetailsTrainer;
  kurse?: EntryDetailsKurs[];
}
```

Abbildung 16 Interface für Abfrage eines Klassenbucheintrags

Formelemente für neuen Eintrag

```html
<form [formGroup]="form">
  <mat-card appearance="outlined" class="card">
    <mat-card-title class="mat-card-title">
      <mat-form-field appearance="fill">
        <mat-label>Einsatz Bezeichnung</mat-label>
        <input type="text"
          matInput
          id="einsatzBezeichnung"
          formControlName="einsatzBezeichnung">
      </mat-form-field>
    </mat-card-title>
    <mat-card-subtitle>{{ this.currentDate | date: 'd MMM y, hh:mm:ss'}}</mat-card-subtitle>
    <mat-card-content class="mat-card-content">
      <mat-form-field appearance="fill">
        <mat-label>Modul Bezeichnung</mat-label>
        <input type="text"
          matInput
          id="modulBezeichnung"
          formControlName="modulBezeichnung">
      </mat-form-field>
      <mat-form-field appearance="fill">
        <mat-label>Inhalte</mat-label>
        <textarea name="klassenbuchEintraegeInhalt"
          id="klassenbuchEintraegeInhalt"
          cols="30" rows="10"
          matInput
          formControlName="klassenbuchEintraegeInhalt"></textarea>
      </mat-form-field>
    </mat-card-content>
    <mat-card-actions>
      <button mat-flat-button
        color="accent"
        (click)="onSubmit()">übernehmen</button>
    </mat-card-actions>
  </mat-card>
</form>
```

Abbildung 17 HTML Formular für neuen Eintrag

```
this.form = this.formBuilder.group({
  einsatzBezeichnung: [null, Validators.required],
  modulBezeichnung: [null, Validators.required],
  klassenbuchEintraegeInhalt: [null, Validators.required],
  kurse: [null, Validators.required]
})
```

Abbildung 18 Formfelder für neuen Eintrag

Abbildungsverzeichnis

Abbildung 1 Beispiel Angular Material Expansion Panel 4
Abbildung 2 Implementation von MSAL in app.component.ts 5
Abbildung 3 Router-Outlet in app.component.html 6
Abbildung 4 Beispiel einer URL Route 6
Abbildung 5 Angular Router Code-Snippet 7
Abbildung 6 Listenansicht für alle Einträge 8
Abbildung 7 Ansicht für einen neuen Eintrag (alle Standorte, 2 Kurse selektiert) 10
Abbildung 8 Ansicht für einen neuen Eintrag (filter nach Standort Karlsruhe, kein Kurs selektiert) 10
Abbildung 9 Detailansicht im Bearbeitungsmodus 11
Abbildung 10 Detailansicht mit aufgeklapptem Kurs und Teilnehmer 12
Abbildung 11 Code-Snippet Register-Entry-Service Get-Requests 13
Abbildung 12 Code-Snippet Register-Entry-Service Post-Requests 13
Abbildung 13 Zeitplanung als Gantt-Diagram (Zeitverlauf in Stunden) 15
Abbildung 14 Interface für neuen Klassenbucheintrag 15
Abbildung 15 Interface für Abfrage der Standorte 15
Abbildung 16 Interface für Abfrage eines Klassenbucheintrags 16
Abbildung 17 HTML Formular für neuen Eintrag 16
Abbildung 18 Formfelder für neuen Eintrag 17