**Systemdokumentation - fachbereater**

|  |  |
| --- | --- |
| Allgemeine Angaben | In unserem Fall |
| Projektleiter | Walden-Quirxtner Astrid |
| Autoren | Raphael Ablinger, Christoph Handel |
| Datum | 15.07.2024 |
| Zeitrahmen Projekt | 05/24 – |
| Versionsnummer | 0.1 |

**INHALT**

[1. Einleitung 3](#_Toc172553636)

[2. Systemübersicht 4](#_Toc172553637)

[2.1.1 Systemarchitektur 4](#_Toc172553638)

[2.1.2 Komponenten – warum wir sie verwenden 4](#_Toc172553639)

[2.1.3 Berechtigungsgruppen 5](#_Toc172553640)

[3. Oberflächen Beschreibung 6](#_Toc172553641)

[3.1.1 Kalender 6](#_Toc172553642)

[3.1.2 Übersicht Anforderungen und Anmeldungen 6](#_Toc172553643)

[3.1.3 Fachberater Anforderung 6](#_Toc172553644)

[3.1.4 Besucheranmeldung 7](#_Toc172553645)

[3.1.5 Seminaranmeldung 7](#_Toc172553646)

[3.1.6 Übersicht Abschlussbericht 7](#_Toc172553647)

[3.1.7 Benutzerverwaltung 7](#_Toc172553648)

[4. Software-Komponenten 7](#_Toc172553649)

[4.1.1 Hibernate Panache 7](#_Toc172553650)

[4.1.2 Resteasy-Jackson 9](#_Toc172553651)

[4.1.3 Keycloak 10](#_Toc172553652)

[5. Deployment 11](#_Toc172553653)

[5.1.1 GitHub Action 11](#_Toc172553654)

[5.1.2 Docker Compose 12](#_Toc172553655)

[6. Entwicklerhandbuch 13](#_Toc172553656)

[6.1.1 Verzeichnisstruktur 13](#_Toc172553657)

[6.1.2 APIs und Schnittstellen 14](#_Toc172553658)

# Einleitung

Weltweit sind bei Almi Vertreter sowie Fachberater unterwegs um jegliche Aufgabenstellung sowie Probleme zu lösen. Da diese Arbeiten natürlich alle koordiniert sowie hinterlegt werden müssen, wurde dies bisher mit etwaigen Pdf-Formularen gemacht.

Das neue Fachberater Programm, dient somit nicht nur zur besseren Koordinierung, ebenso erleichtert es die Eingabe sowie Benutzerverwaltung der für Almi tätigen Vertreter sowie Fachberater.

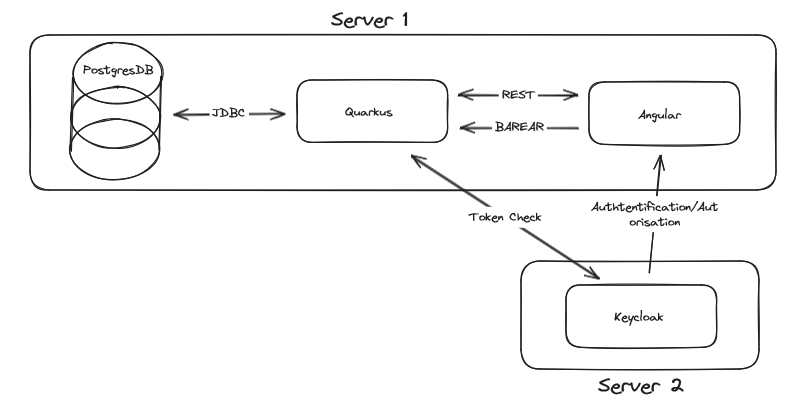
Der Zweck dieser Dokumentation ist es, Entwicklern zu ermöglichen, sich schnell und effizient in das Programm einzuarbeiten und es weiterzuentwickeln. Diese Dokumentation richtet sich daher vor allem an die Software Entwickler der Firma Almi.

# Systemübersicht

Um in den folgenden Abschnitten besser erklären zu können, wie die einzelnen Komponenten eingesetzt worden sind, sind hier die einzelnen Oberflächen beschrieben. Alle Schlüsselelemente sowie die wichtigsten Buttons sind mit Berechtigungsgruppen ausgestattet.

Diese Berechtigungsgruppen werden mit Keycloak gesteuert und verwendet, um auf den Oberflächen nur das anzuzeigen, was der jeweilige Benutzer auch sehen darf. Die Hauptfunktionen werden kurz aufgelistet.

### Systemarchitektur



### Komponenten – warum wir sie verwenden

Quarkus**:**

* **Schnelle Startzeiten:** Quarkus ermöglicht, sehr schnelle Startzeiten und einen geringen Speicherverbrauch deswegen ist Quarkus für das Fachberater Programm ein sehr hilfreiches Framework.
* **Effizienter Ressourcenverbrauch:** Durch die Optimierung für Containerumgebungen wird der Ressourcenverbrauch minimiert, was zu Kosteneinsparungen und besserer Leistung führt.
* **Entwicklerfreundlichkeit:** Da Quarkus von Grund auf eine Vielzahl an Erweiterungen bietet und mit den meisten gängigen Java-Standards und -Bibliotheken kompatibel ist, beschleunigt sowie vereinfacht Quarkus dadurch die Entwicklung enorm.

**Angular:**

* **Modularität:** Angular ermöglicht eine modulare Entwicklung, wodurch der Code besser organisiert und wartbar bleibt. Die verschiedenen Oberflächen können dadurch ordentlich in einzelne Folder aufgeteilt werden.
* **Interaktive Benutzeroberflächen:** Angular eignet sich hervorragend zur Erstellung dynamischer und interaktiver Benutzeroberflächen, somit ist Angular natürlich perfekt für das Fachberater Programm geeignet
* **Zukunftssicher:** Ebenso ist Angular ein sehr modernes/zukunftssicheres Framework was natürlich bei so einem Programm sehr wichtig ist. Die gute Testbarkeit war ebenso entscheidend bei der Wahl dieses Frameworks.

**Keycloak:**

* **Zentrale Benutzerverwaltung:** Keycloak ermöglicht eine zentrale Verwaltung von Benutzern und Berechtigungen, was die Sicherheitsadministration vereinfacht.
* **Rollenbasierte Zugriffskontrolle:** Wie schon bereits beschrieben werden die berechtigungsgruppen mit Keycloak verwaltet. Die Zuweisung von Benutzer-/ Gruppenrollen und Berechtigungen auf Basis der spezifischen Anforderungen der Anwendung ist somit enorm vereinfacht.
* **Admin Oberfläche:** Mit Hilfe der Admin Oberfläche kann man einfach neuer User, Groups, Roles, … anlegen bzw. verwalten. Ebenso kann man laufende Sessions (Benutzer die zurzeit die Oberflächen benutzen) beobachten.

### Berechtigungsgruppen

Um zuweisen zu können wer auf welche Oberflächen, Elemente, Buttons, … zugreifen darf, wurden Berechtigungsgruppen verwendet.

Vorgeben wurde sieben verschieden Gruppen (7=Admin):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GL + Assistenz** | **Abteilungsleiter** | **Vertreter** | **Fachberater** | **Front Office** | **Händler/Tochter** |

Diese wurden von uns in Zahlen von 1-7 eingeteilt. Diese Berechtigungsgruppen wurden absteigend nummeriert wobei 1 die mächtigste Rolle ist und 6 die Rolle mit den wenigsten Rechten ist.

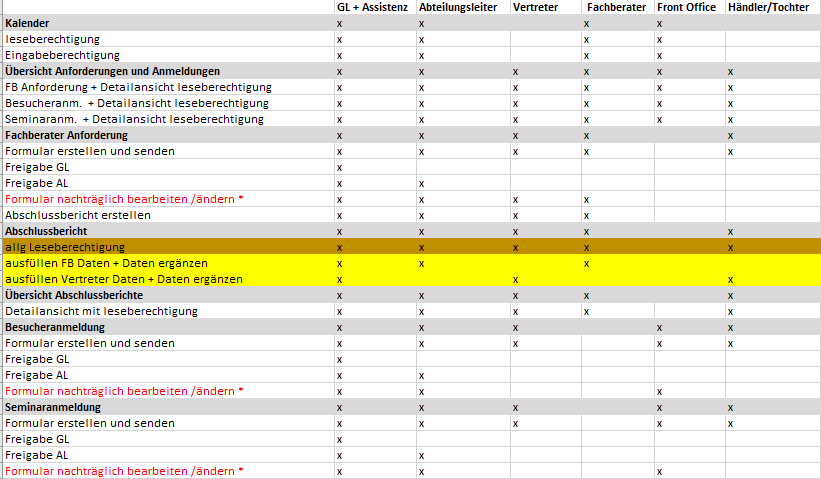
Die Rolle mit der Nummer 7 ist die Admin Rolle, die logischerweise alle Rechte besitzt und somit den Zugriff auf den gesamten Inhalt der einzelnen Oberflächen hat.

\*ngIf="roleService.checkPermission([1,2,7])"

🡪 mit roleService.checkPermissions wird der Zugriff auf einzelne Elemente kontrolliert

{ path: 'mainList', component: MainListComponent, canActivate: [authGuard], data: { requiredRoles: [1, 2, 3, 4, 5, 6, 7] } },

🡪 im app-routing.module.ts wird mit Hilfe von „requiredRoles: []“ angeben, auf welche Oberflächen mit der jeweiligen Berechtigung geroutet werden darf



# Oberflächen Beschreibung

### Kalender

Mit Hilfe des Kalenders können die Berechtigten User die Fachberater einteilen, so wie auf einen Blick sofort sehen, welche Fachberater verfügbar/verplant sind.

### Übersicht Anforderungen und Anmeldungen

Auf dieser Oberfläche sind die drei verschiedenen Formulare: Besucheranmeldung/Seminaranmeldung/Fachberater Anforderung in einer Liste sichtbar. Diese Liste ist mit Such-/ sowie Filterfunktionen ausgestattet. Mithilfe des Buttons am Ende jedes Eintrags kann – wenn Berechtigungen ausreichend- , der bestimmte Eintrag unsichtbar gesetzt werden.

### Fachberater Anforderung

Auf der Oberfläche Fachberater Anforderung, können geplante Kundenbesuche eingetragen werden ebenso kann pro geplanten Kundenbesuch ein neuer Abschlussbericht erstellt werden.

Es können beliebig viele Kundenbesuche angelegt werden.

### Besucheranmeldung

Um Besucher anzumelden wurde die Oberfläche Besucheranmeldung erstellt. Entweder wird ein Aufenthalt in Oftering, ein Kundenbesuch, ein Almi Mitarbeiter oder alle drei Anmeldung benötigt. Dies wird natürlich auch überprüft sodass es nicht vorkommt das eine Besucheranmeldung ohne jegliche Daten angemeldet werden kann. Weiter Anforderungen wie: eine Betriebsführung, den Besprechungsraum, Mittagessen, … können unter Angabe verschiedenstet Parameter hinzugefügt werden.

Um Almi darüber zu informieren wie viele und vor allem welche Teilnehmer Almi besuchen wurde ein eigener Dialog implementiert.

### Seminaranmeldung

Ähnlich wie die Besucheranmeldung können mit der Oberfläche Seminaranmeldung ein neues Seminar angelegt werden. Wie auf jeder Oberfläche müssen auch bei dieser verschiedene Pflichtfelder ausgefüllt werde, die detaillierte Teilnehmerliste ist ebenso ein Pflichtfeld. Weiter Anforderungen für ein Seminar können unter anderem zum Beispiel: eine Hotelbuchung, ein Mittagessen, ein Ausflug, … sein.

### Übersicht Abschlussbericht

Bei dieser Oberfläche werden alle Abschlussberichte in einer Liste angeführt. Mit den verschieden Filter Möglichkeiten wird die Suche nach den Abschlussberichte erleichtert. Um die Liste übersichtlicher zu machen werden bei den Artikeln nur die ersten zwei Artikelnummern angezeigt. Mit Hilfe eines Buttons können alle Artikelnummern angezeigt werden.

### Benutzerverwaltung

Um neue Fachberater/Vertreter/Händler hinzuzufügen oder bearbeiten zu können wurde die Oberfläche Benutzerverwaltung erstellt. Diese Oberfläche wurde in drei Unterseiten unterteilt:

* Neuer Fachberater
* Neuer Vertreter
* Neuer Händler.

Auf der jeweils der linken Seite der Oberfläche können die Namen bzw. die Bezeichnung oder Farben eingetragen werden. Die rechte Seite des Bildschirms beinhaltet eine Liste mit den bereits angelegten Datensätzen.

Mit Hilfe eines Klicks können bereits bestehende Datensätze bearbeitet werden.

# Software-Komponenten

### Hibernate Panache

Mithilfe von Hibernate Panache wird das vereinfachte persistieren von Daten in Datenbanken ermöglicht. Hierzu wird eine @Entity Annotation über dem Entity und ein default Constructor benötigt.

@Entity *public class* Car *extends* PanacheEntity{  
 *public int horsepower;*

*public string brand;*

*…*

*public Car(){}*  
}

Objekte können nach dieser Annotation ganz einfach mit der Methode *.persist();* in der Datenbank gespeichert werden.

*Car car1 = new Car()*

*car1.horsepower = 52;*

*car1.brand = “Daihatsu”*

*car1.persist();*

Wichtig die Methode, in der *.persist();* aufgerufen wird, muss mit @Transactional markiert sein.

@Transactional bewirkt, dass alles was in dieser Methode aufgerufen wird in einer Transaktion abgearbeitet wird.

Die Daten können mit der Methode *.find()* aus der Datenbank entnehmen. Hierzu werden mindestens 2 Parameter gefordert. *.find(<Name im Entity>, <value>)*. Wenn ein Entity mit einem *.find()*  aus der Datenbank entnommen wird ist der Status dieses Entities persistent, das bedeutet, wenn im Programm etwas umgeändert wird, aktualisiert es sich somit auch automatisch in der Datenbank. Dies ist nicht nur bei einem *.find(),* sondern auch bei einem *.persist()* und noch weiteren Panache Methoden der Fall.

*Car modifyCar = Car.find(“brand”, “Daihatsu”);*

*modifyCar.horsepower = 100;*

*Car moreDetailed = Car.find(„brand = ?1 and horsepower > ?2”, “Alpha Romeo”, “70”);*

Wenn nach mehreren Feldern gesucht wird kann mit ?1, ?2, ?... bestimmt werden welche Variable verwendet wird.

Ebenso kann man Vererbungen damit abbilden.

Hierzu muss:

1. Das vererbte Objekt selbst mit @Entity markiert sein.
2. In der Parentklasse mit einer Annotation:
   1. @ManyToOne: Wenn das Vererbte Entity mehrere Parent Entities haben kann
   2. @OneToMany: Wenn das Parent Entity mehrere vererbte Entities haben kann
   3. @OneToOne: Wenn jeweils nur eins vorhanden sein kann
   4. @ManyToMany: Viele zu Viele Relation

Beispiel:

@Entity  
*public class* Car *extends* PanacheEntity {  
 …

@OneToMany  
 *public List*<Company> brand;

…  
}

Hier wird @OneToMany verwendet da Car (sprich das Parent Entity) mehrerer Company Entities (sprich vererbte Entities) haben kann.

Wiederum wenn man sich diese Beziehung von der anderen Seite ansehen würde, würde ein @ManyToOne benötigt werden.

Vorsicht: Vererbungen sollten nur von einer Seite aus gemacht werden, da es sonst zu Fehlern kommen kann.

User Aufbau:

In jedem unserer Entities haben wir die gleiche Struktur für das Persistieren und Updaten. Darin befindet sich die persistOrUpdate() und die update() Methode.

persistOrUpdate() Methode

Diese Methode überprüft ob das übergebene Objekt bereits persistiert ist. Es wird überprüft ob es bereits eine ID zugewiesen bekommen hat oder nicht. Falls nicht wird es direkt persistiert um einen persistenten Verbindungszustand zwischen Datenbank und Applikation zu schaffen. Danach wird bei allen Beziehungen/Vererbungen selbst persistOrUpdate() ausgeführt, sodass alle Beziehungen aktualisiert werden.

Wenn ein Entity bereits persistiert ist, so wird als erstes das persistierte Objekt aus der Datenbank gesucht und anschließend vom persistierten Objekt mit der Methode update() geupdatet und mit dem aktuellen Objekt aufgerufen.

Die update() Methode überschreibt alle vorhandenen Felder mit den neuen Feldern.

Diese Felder werden dank des Zustandes „persistiert“ direkt in die Datenbank gespeichert.

### Resteasy-Jackson

Die Library Resteasy-Jackson ist zuständig für die Restverbindungen und für das automatische Mappen zwischen Json und Java Entities.

Diese Library erleichtert das Bereitstellen von Schnittstellen.

Beispiel einer Schnittstelle:

@GET

@Path(“/sample”) *public* Response getAllTechnologists(){  
 *return* Response.*ok*(Technologist.*listAll*()).build();  
}

Das @Get gibt an um welche CRUD Methode es sich hierbei handelt.

Mit @Path() wird angegeben unter welchem Pfad man die Schnittstelle findet.

Mit der Methode Response w wird ein beliebiges Objekt zurückgegeben.

Man kann auch Parameter wie @PathParam(<value>)oder @QueryParam(<value>)verwernden um spezielle Objekte zu erhalten. Hierfür wird einfach im Methodenkopf als Parameter

public Response getPdf(@PathParam("text") String text)

Bei einem @Post muss das Entity das von der Schnittstelle kommt ebenso als Parameter im Methodenkopf angebenden werden.

Es ist jedoch wichtig, dass gleich bei einem @Get oder @Post, im frontend und im backend in den Entity Felder immer gleich benannt sein müssen, ansonst kann Jackson sie nicht mappen.

@Path("technologist")  
*public class* TechnologistResource {

Ebenso wichtig ist es, in den Resourcklassen über die Klasse ein @Path() zuschreiben.

Somit weiß Resteasy, dass dies eine Schnittstellenklasse ist.

Wenn über einer Klasse zum Beispiel "technologist" steht und über der Resource Methode "sample" muss die Methode dann im frontend mit <http://host:port/technologist/sample> aufgerufen werden.

### Keycloak

Es wurde ein Realm namens „fachbeate“ erstellt. In den Realm Settings wurde ein eigenes Almi\_theme erstellt -> dieses Theme verwendet Bilder sowie CSS Files, die auf dem Keycloak Server bereitgestellt werden, um die Login Page zu personalisieren. Mit Hilfe eines Docker Volumes wird das Theme für das Docker Image bereitgestellt.

Ebenso war der Wunsch, dass das automatische Logout erst nach 1-Stunde erfolgt -> dies wurde unter dem Reiter Sessions eingestellt.

**Clients**

Beifolgenden Clients wurden Root-, Valid redirect-, Web origins- sowie Admin URLs gesetzt:

* **fachbeate-backend**
* **fachbeate-backend-dev**
* **fachbeate-frontend**
* **fachbeate-backend-dev**

In fachbeate-frontend wurden die verschiedenen Rollen die bereits beschrieben worden unter dem Reiter „Roles“ hinzugefügt.

**Users**

Bei dem Reiter „Users“ müssen alle Accounts die mit dem Fachberater Programm arbeiten, angelegt werden.

WICHTIG

Bei den Händlern/Töchtern 🡪 muss der Username exakt mit dem Username, der in der Applikation angegeben wird, übereinstimmen.

Bei den Vertretern/Fachberatern 🡪 gilt dasselbe, nur muss hier die E-Mail übereinstimmen

**Groups**

Für jede Berechtigungsebene wurde eine Gruppe erstellt, dies erleichtert die den Autorisierungsprozess enorm, da somit nicht jeden einzelnen User eine/mehrere Rollen zugewiesen werden muss, sondern nur eine Gruppe.

**Sessions**

Unter dem Reiter „Sessions“ kann man die User sehen, die aktuell mit der Applikation verbunden sind.

# Deployment

### GitHub**Action**

Mithilfe der GitHub Actions ist das Automatisierte Bauen von Docker Images von unseren Applikationen möglich. Die „Bauanleitung“ bzw. die GitHub Action liegt im Verzeichnis: /.github/workflows/ci.yml.

Diese „Bauanleitung“ beinhaltet zwei sogenannte „Jobs“:

* build-backend
* build-frontend.

Wie der Name schon sagt ist build-backend für das Backend zuständig und build-frontend für das Frontend.

Um die GitHub Actions auszuführen, müssen jeweils die unten angeführten Schritte ausgeführt werden.

**Build-backend**

1. uses: actions/checkout@v3

* zieht sich das Repository in die “virtuelle Maschine” in der gh-action

1. Build with Maven:

* Gibt dem maven building tool alle benötigten Rechte und packaged es dann

1. Setup with QEMU

* Ist dazu notwendig um die Applikation für verschiedene CPU-Architekturen lauffähig zu machen

1. Convert GitHub repository name to lowercase

* Erstellt eine Variable von dem Namen der gh-registry
  + gh-registry: Ort wo später alle Docker Images gespeichert werden

1. Set up Docker Buildx

* Action um Docker auf die „virtuelle Maschine“ zu installieren

1. Login to GitHub Container Registry

* Verwendet die schon vorhandenen GitHub-secrets um sich in die gh-registry einzuloggen um später die Images hochzuladen

1. Build and push Docker image

* Verwendet das Dockerfile welches im backend liegt um ein Docker Image zu erstellen. Noch dazu wird überprüft ob Libraries oder andere Teile schon bereits einmal gebuildet wurden. Wenn dies der Fall ist werden diese gechached und beim nächsten build wieder verwendet.

Build-frontend

Ident mit dem Build-backend Prozess nur das kein Java installiert werden muss.

Wie oben bereits beschrieben werden alle fertig gebauten Docker-Images in die gh-registry gespeichert.

Mit dem Command „docker run ghcr.io/qmchri/fachbeate-api:latest“ startet man den Docker Container.

Die GitHub Action wird immer mit einen push auf den main-Branch ausgeführt. Den aktuellen Fortschritt einer Action kann man im Repository unter dem Reiter: Action ansehen.

Wenn beide Häkchen grün sind, sind die neuersten Versionen auf der gh-registry

### Docker Compose

Um die neueste Version auf den Server zu pushen ist es notwendig, die Images auf dem Server erstens neu zu pullen und dann neu zu starten.

Dafür wurde das „docker-compose.yml“ File erstellt.

Im Grunde ist dieses File nur eine Anleitung welche Images von

* wo entnommen werden und
* mit was sie gestartet werden müssen.

Solche docker-compose Files findet man:

* zweimal am Angular Server (angular02.almi.at) und
* einmal am Keycloak Server (keycloak01.almi.at).

Das wichtigste docker-compose ist jedoch das eine am Angular Server im Ordner angular. Dieses File regelt die Quarkus und die Angular Applikation. Sprich wenn eine neue Version vorhanden ist, muss über dieses docker-compose File die Applikationen aktualisiert werden.

dazu:

1. wechselt man in den angular Ordner
2. Stoppt die aktuell laufenden Container mit: „sudo docker compose down“
3. Pulled die neuen Images von GitHub mit: „sudo docker compose pull“
4. Danach startet man die Container neu mit: „sudo docker compose up -d“
   1. Hierbei steht das -d für detached, sprich man sieht keine Logs von den Containern, denn ansonsten würden man automatisch die Container stoppen sobald man das Terminal verlässt
   2. Man kann den Befehl ohne -d verwenden, wenn man debuggen möchte

Wenn man die Container detached laufen hat, möchte aber trotzdem die Logs sehen, da ein Fehler aufgetreten ist, so müssen folgende Schritte durchgeführt werden:

1. Mit “sudo docker ps” sieht man alle laufenden Container -> somit kann man sich die ID von den Containern wo der Fehler aufgetreten ist kopieren.
2. Danach können mit „sudo docker logs <ID vom Container>“ die Logs und Container angesehen werden.

Wenn Änderungen bei den Quarkus Entities gemacht wurden, so muss auch die Datenbank an die Änderungen angepasst werden.

Hierzu muss im docker-compose bei Quarkus folgende „env“ Variable gesetzt werden

*quarkus.hibernate-orm.database.generation=drop-and-create*

ACHTUNG:

Bei diesem Command werden alle Daten aus der Datenbank gelöscht

also muss unbedingt eine Zwischenkopie von der Datenbank erstellt werden sodass im Nachhinein die Daten wieder zurückgespeichert werden können!

# Entwicklerhandbuch

Es wurden in der Applikation die wichtigsten Zeilen Code mit Kommentaren versehen, um die Dokumentation somit etwas zu vereinfachen. In dieser Dokumentation wurden nur die Verzeichnisstrukturen sowie alle Schnittstellen des Programms aufgeführt um einen Überblick über die Applikation zu bekommen.

### **Verzeichnisstruktur**

**Frontend**

Unterverzeichnis: src/app

 **app-routing.module.ts**: Datei, die die Routen für die Anwendung definiert.

 **app.component.css**: CSS-Datei für das Styling des Hauptkomponenten.

 **app.component.html**: HTML-Datei, die das Template für die Hauptkomponente enthält.

 **app.component.spec.ts**: Testdatei für die Hauptkomponente.

 **app.component.ts**: TypeScript-Datei, die die Logik der Hauptkomponente implementiert.

 **app.module.ts**: Hauptmodul der Angular-Anwendung, dass die Abhängigkeiten und Komponenten importiert und deklariert.

Unterverzeichnis: src/app/components/sidenav

 **helper.ts**: interface für die Sidenav

 **nav-data.ts**: Routen für die einzelnen Komponenten, direkt im Code - genauer beschrieben.

 **initializer.service.ts**: Service zur Initialisierung des Keycloakservices.

Unterverzeichnis: models

Hier sind alle interfaces/Entities lokalisiert

Unterverzeichnis: services

 **auth.service.ts**: Service für die Authentifizierung.

 **http.service.ts**: Service für HTTP-Anfragen.

 **notification.service.ts**: Service für Benachrichtigungen.

 **role.service.ts**: Service für die Verwaltung von Rollen.

Unterverzeichnis: assets/i18n

* **de.json**: Deutsche Übersetzungen.
* **en.json**: Englische Übersetzungen.
* **ru.json**: Russische Übersetzungen.

Unterverzeichnis: environments

* **environment**.**development**.**ts:** Entwicklungsumgebungsvariablen für lokale Entwicklung**.**
* **environment.ts:** Entwicklungsumgebungsvariablen für das Development**.**
* **index**.**html:** Haupt-HTML-Datei der Anwendung.
* **main**.**ts:** Haupteinstiegspunkt für die Angular-Anwendung.
* **styles**.**scss:** Hauptstylesheet für die Anwendung.

**Backend**

Unterverzeichnis: src/main/java

* **boundary:** alle Schnittstellen des Backends.
* **entity/dto:** alle Entities sowie DTOs.

Unterverzeichnis: src/main/resources

* **application.properties:** Environment File

Unterverzeichnis: src

* **mvnw:** MavenWrapper**:** BuildingTool
* **pom.xml:** File in der alle Dependencies definiert sind

### **APIs und Schnittstellen**

Workshop (Alle Seminar Schnittstellen)

* **GET /workshop**
* **POST /workshop**
* **GET /workshop/id**
* **GET /workshop/user**

Appointment (Alle anderen Schnittstellen die man für jegliche Anforderungen Benötigt)

* **GET appointment/article**
* **GET appointment/finalReport**
* **POST appointment/finalReport**
* **GET appointment/finalReportByUser**
* **GET appointment/other**
* **POST appointment/other**
* **GET appointment/other/id**
* **GET appointment/other/user**
* **GET appointment/visibility**

customerRequirement (Alle Fachberater Anforderung Schnittstellen)

* **GET customerRequirement**
* **POST customerRequirement**
* **GET customerRequirement/id**
* **GET customerRequirement/user**

Users (Schnittstellen um alle Benutzer zu bekommen)

* **GET**users/company
* **POST**users/company
* **GET**users/company/allActive
* **GET**users/representative
* **POST**users/representative
* **GET**users/representative/allActive
* **GET**users/technologist
* **POST**users/technologist
* **GET**users/technologist/allActive

visitorRegistration (Alle Kundenbesuch Schnittstellen)

* **GET visitorRegistration**
* **POST visitorRegistration**
* **GET visitorRegistration/id**
* **GET visitorRegistration/user**