DOKUMENTATION ZUR BETRIEBLICHEN PROJEKTARBEIT

SYNCRONISATION VON ROLLENZUORDNUNG ZWISCHEN ENTWICKLUNGSUMGEBUNG UND PRODUKTIVUMGEBUNG

Christoph Schneekloth | Fachinformatik Anwendungsentwicklung | 2023

Inhaltsverzeichnis

1. Projekteinführung

1.1 Projektumfeld

Der Auftraggeber ist die Technische Universität Hansestadt Hamburg (TUHH). Die TUHH wurde 1978 gegründet und hat rund 7.900 Studierende, über 100 Professorinnen und Professoren sowie rund 1.200 Mitarbeiterinnen und Mitarbeiter. Die TUHH hat ihr eigenes Rechenzentrum mit eigener IT, welche es den Studenten ermöglicht, ihren Lehr- und Vorlesungsalltag zu vereinfachen. Außerdem verwalten sie ebenfalls die Homepage der TUHH.

1.2 Projektumsetzung

Die Projektumsetzung fand hauptsächlich bei dem Auszubildenden Christoph Schneekloth im Home-Office statt. Auftraggeber und Hauptansprechpartner war Jakov Jakus. Sämtliche Programmiertätigkeiten sowie Tests und Anpassung wurden vom Auszubildenden für Anwendungsentwicklung Christoph Schneekloth durchgeführt.

1.3 Projektziel

In diesem Projekt soll eine CRUD-Anwendung entwickelt werden, die den Administrator erlauben, die Rollen für eine Person wieder manuell nachzutragen. Dieses Ziel erfordert eine neu entwickelte Software mit simpler Benutzeroberfläche und entsprechender Auswahl- und Bearbeitungsmöglichkeit für den Benutzer. Der Administrator sollte dann die Rollen einfach nachtragen können. Die Ausganglage betrifft

2. Projektanalyse

2.1 Ist-Analyse

Die Server der TUHH werden intern als Säule bezeichnet. In der TUHH gibt es Säulen für das Intranet, externen- und internen Zugriff, Ersatz-Server und Test-Server. Jede Säule hat eine bestimmte Aufgabe. Die Konfigurationen und Software von diesen Säulen werden bei Aktualisierungen dann auf die anderen Säulen übertragen. Mitarbeitern der TUHH ist aufgefallen, dass sich die Rollen von den Einträgen in der Produktivsäule und der Entwicklungssäule unterscheiden. Mit der Aktualisierung der Datenbank auf der Entwicklungssäule würden diese Rollen zwangsläufig auf der Produktivsäule verloren gehen.

2.2 Soll-Analyse

Mit der Anwendung sollen Administratoren es schnell und einfach haben, Rollen manuell nachzutragen. Hierzu soll die Programmoberfläche die Möglichkeit bieten, aus einer Liste aller Rollen und Säulen in einem Formular die benötigten auszuwählen und per Klick diese zu übernehmen. Das fertige Formular soll abschließend gespeichert und abgesendet werden, um die Rollen und Säulen eines Mitarbeiters zu aktualisieren.

3. Projektplanung

3.1 Projektphasen

Dem Autor standen laut IHK-Vorgaben 80 Stunden Arbeitszeit zur Verfügung. Die Durchführung des Projektes fand in der regulären Tagesarbeitszeit statt. Die Stunden wurden über diesen Zeitraum verteilt:

Projektphase	Geplante Dauer in Stunden
Projektanalyse	4
Wirtschaftlichkeit	4
Datenbankerstellung	5
Backendentwicklung	20
Frontendentwicklung	25
Qualitätssicherung	10
Projektdokumentation	12

Tabelle 1: Geplante Stunden pro Projektphase

3.2 Ressourcenplanung

Zur Entwicklung wurde eine Java- und JavaScript fähige Entwicklungsplattform benötigte, verwendet wurden IntelliJ und Visual Studio Code. In IntelliJ benutzte man das Framework Spring Version 3.1.2 und ein paar Plug-Ins. Eins der Plug-Ins war Lombok. Lombok automatisiert die Erzeugung von Code und reduziert den Boilerplate-Code, den Entwickler normalerweise schreiben müssen. Außerdem generiert Lombok Getter, Setter und Konstruktoren, wodurch der Entwickler Zeit spart und der Code wird lesbarer und wartungsfreundlicher. Deshalb fiel meine Wahl auf Lombok, um effizienter zu arbeiten. Für die API wurde Java Persistence API (JPA) verwendet. JPA wird zum Zugriff auf die Datenbank und für die Speicherung und Abrufung von Objekten in den Daten.

Für die Frontentwicklung auf Visual Studio Code wurde sich für JavaScript und das Node.js Framework. Zum Node.js Framework wurde dann noch React implementiert, um eine schnelle und simple Benutzeroberfläche zu entwickeln. Man entschied sich für Reactstrap als Bibliothek, da man eine einfache, aber trotzdem schöne GUI wollten. Reactstrap ist eine Bibliothek für React, die React Bootstrap-Komponenten enthält. Im Gegensatz zur Verwendung von Bootstrap in HTML exportiert Reactstrap automatisch alle korrekten Bootstrap-Klassen und erfordert nicht die Verwendung oder Einbindung von Bootstrap's JavaScript. JavaScript wurden aus dem Grund gewählt, da es momentan der Standard ist, wenn es um Frontend geht. Die Anwendung sollte zeitgenössisch sein und die Entwicklung möglichst schnell und einfach machen.

Die Datenbank zum Protokollieren der Transaktionen und temporäre Speicherung der Daten wurden über PostGreSQL und pgAdmin4 realisiert. Bei der verwendeten Software und den Plug-Ins sowie Frameworks wurde darauf geachtet, dass diese kostenlos zur Verfügung stehen bzw. sowieso schon im Betrieb lizenziert sind oder eingesetzt werden. So entstanden dem Ausbildungsbetrieb durch die Entwicklung der Software keine zusätzlichen Kosten.

3.3 Zielplattform

Als Zielplattform wurde Microsoft Windows gewählt, da der PC des Administrators auf Windows 10 betrieben wird. Deshalb musste keine Rücksicht auf andere Betriebssystem genommen werden. Als Programmiersprachen wurde sich für Java für Backend und JavaScript für Frontend entschieden, da diese bei Programmiertätigkeiten in der TUHH vorwiegend eingesetzt wird. Außerdem können mit Java und JavaScript alle erforderlichen und gewünschten Ziele umgesetzt werden.

3.4 Architekturdesign

Das Projekt basiert auf dem Architekturprinzip MVC (Model-View-Controller). Die Software lässt sich gemäß diesem Muster in die drei Bereiche Model (Datenaufbereitung), View (Präsentation) sowie Controller (Anwendungssteuerung) unterteilen. Aus dieser Trennung ergeben sich einige Vorteile wie bessere Wartbarkeit, Lesbarkeit und Erweiterbarkeit. Da die Logik von der Darstellung getrennt ist, kann man etwa die Oberfläche leicht austauschen, ohne tiefer in den logischen Bereich der Software vordringen zu müssen.

4. Projektentwicklung

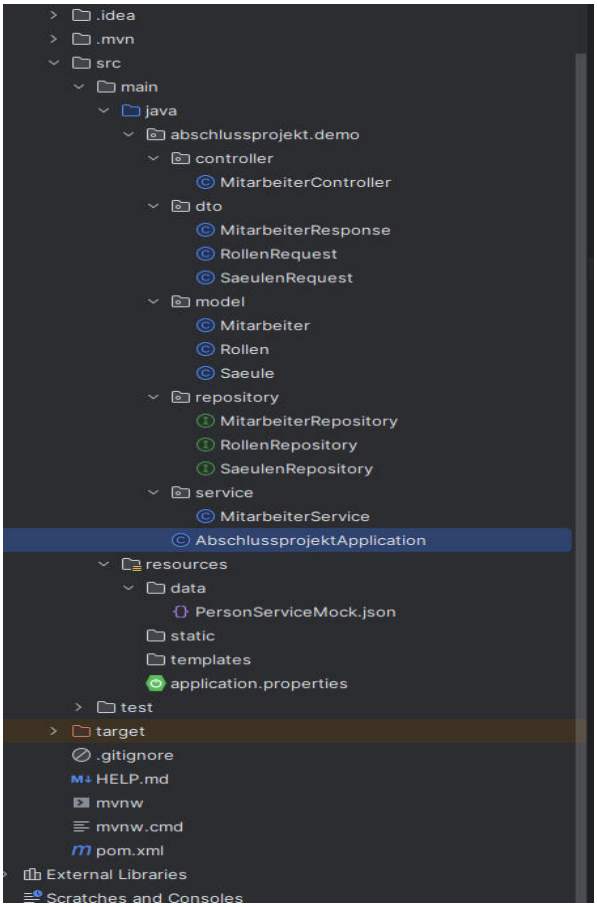
4.1 Erstellung des Backend Ordner

Spring Boot Projekte können manuell eingerichtet werden, aber die Konfiguration kann zeitaufwendig und verwirrend sein. Zur Erstellung des Spring Boot Projektordners wurde sich für den Spring Initializr entschieden. Spring Initializr ist eine Website oder ein webbasiertes Tool, das verwendet wird, um Spring Boot Projekte auf einfachere und effizientere Weise einzurichten. Der Spring Initializr bietet 3 Programmiersprachen zur Auswahl: Java, Kotlin oder Groovy. Da sich das Projekt aber mit Java realisieren lassen soll, war die Auswahl nicht schwer. Ebenfalls kann man entscheiden, welches Build-Management-Tool man benutzt, wo man sich für das Spring Boot Maven Plug-In entschieden hat.

Das Spring Boot Maven Plug-In ist ein Plug-In für das Build-Management-Tool Maven, das speziell für die Entwicklung von Spring Boot-Anwendungen entwickelt wurde. Das Plug-In ermöglicht es , ausführbare JAR- oder WAR-Archive zu erstellen, in denen deine Spring Boot-Anwendung verpackt ist. Es bietet Unterstützung für das Ausführen von Spring Boot-Anwendungen direkt aus dem Build-Prozess heraus und generiert Build-Informationen, die in der Anwendung verwendet werden können. Das Plug-In ermöglicht außerdem deine Spring Boot-Anwendung vor dem Start zu konfigurieren und anzupassen und bietet auch Funktionen zur Verwaltung von Abhängigkeiten, indem es die Versionen der gemeinsamen Abhängigkeiten verwaltet. Mit dem Spring Boot Maven Plug-In kann man also Spring Boot-Anwendungen effizienter entwickeln, verpacken und ausführen.

Zurück zum Spring Initializr muss man noch die Verpackungs-Art (JAR/WAR) und die Java Version auswählen. Die Entscheidung fiel natürlich auf Java bei der Sprache, da das Projekt in Java realisiert werden soll. Außerdem entschied man sich für JAR, denn mit JAR hatte man bisher die beste Erfahrung gemacht in anderen Entwicklerumgebungen (zum Beispiel Eclipse) gemacht und es lief am besten mit Java, wenn es um verpacken und entpacken von Modulen geht. Danach kommt man zu den Dependencies. Das Spring Framework verwendet das Prinzip der Dependency Injection (DI), bei dem Objekte ihre Abhängigkeiten nicht selbst erstellen, sondern von einer externen Quelle bereitgestellt bekommen. Dadurch wird die Flexibilität und Testbarkeit der Anwendung verbessert. Die Abhängigkeiten im Spring Framework werden in Form von Java-Bibliotheken oder Modulen ausgedrückt, die in dem Projekt eingebunden werden.

Die ausgewählten Dependencies und Plug-Ins waren Lombok(Erklärt in Ressourcenplanung, 3.2), Spring MVC, REST API, Spring Data JPA für das Repository und der PostGreSQL Treiber. Hauptsächlich nahm ich die Spring Plug-Ins (Spring MVC und Spring Data JPA), weil diese am besten mit Java funktionierten.



Projekt Hierarchie Backend

In Entwicklerumgebung IntelliJ musste man nichts vorher konfigurieren, da das alles im Spring Initializr vorkonfiguriert war. Einzige Änderung, die man vornahm, war die Einbindung der Datenbank in der „application.properties“-Datei. Die "application.properties"-Datei ist eine Konfigurationsdatei, die in Spring-Anwendungen verwendet wird, um Einstellungen und Eigenschaften zu definieren. Sie dient dazu, verschiedene Konfigurationseinstellungen für die Spring-Anwendung zu speichern wie zum Beispiel Datenbankverbindungsinformation und den Serverport. Die Verwendung der "application.properties"-Datei erleichtert die Verwaltung von Konfigurationseinstellungen in deiner Spring-Anwendung und ermöglicht eine einfache Anpassung an unterschiedliche Umgebungen.

Die Abhängigkeiten und Plug-Ins werden nach der Generierung des Projektordners in der pom.xml gespeichert. Die "pom.xml"-Datei ist ein zentrales Element in Maven-basierten Java-Projekten und wird verwendet, um Projektmetadaten sowie Konfigurationen für das Build-Management und Abhängigkeitsverwaltung zu definieren. Maven lädt diese Bibliotheken automatisch herunter und bindet sie in das Projekt ein.

4.2 Anwendungsentwicklung mit Java

4.2.1 Erstellung der Klassen „Mitarbeiter“, „Rollen“ und „Saeulen“

Diese drei Klassen beinhalten die Attribute der einzelnen Entitäten in der Datenbank. Alle 3 Klassen werden mit der @Entity-Annotation markiert. Die Entity-Annotation signalisiert der JPA-Schnittstellen, dass es sich bei dieser Klasse um eine JPA-Entität handelt. Das bedeutet, dass die Klasse einer Datenbanktabelle zugeordnet wird.

Diese Klasse erstellt in der Datenbank die Tabelle (zum Beispiel: Mitarbeiter mit all seinen Attributen). In jeder der drei Klassen wurde das Attribut „id“ mit der Primärschlüssel-Annotation versehen. Danach wurde die „main“-Klasse gestartet, um die Tabellen in der Datenbank zu erstellen. Es wurde dann auf pgAdmin 4 getestet. Da die TUHH hauptsächlich mit pgAdmin 4 und PostGreSQL arbeitete, fiel die Entscheidung für ein Datenbank-Management-Programm nicht schwer.

4.2.2 Die Erstellung der Service-Klasse MitarbeiterService und Repository

Es wurde eine Service-Klasse erstellt, welche die gesamte Logik beinhaltet. Eine Spring Service-Klasse ist eine Java-Klasse, die Geschäftslogik Ihrer Anwendung enthält. Sie dient zur Trennung der Geschäftslogik von anderen Schichten wie der Controller-Klasse. Die Controller-Klasse gibt die vom Benutzer eingebenden Daten weiter and die Service-Klasse, die dann mit dem Repository zusammenarbeitet, damit das Repository die Daten über die von der Datenbank holt.

Für die Methode „findeAlleMitarbeiterimExternenSystem()“ hat man sich eine Mock.json erstellt. Mock-Objekte wurden eingesetzt, um das isolierte Testen von einzelnen Objekten zu erleichtern. Die „findeAlleMitarbeiterimExternenSystem()“-Methode konvertiert dann diese Mock.json Daten zu einem Model Objekt. Dazu kam noch die Methode „saveRoleForMitarbeiter“, die dafür sorgte, dass die eigebenen Daten lokal abgespeichert werden und in der Datenbank protokolliert werden. Diese Protokolle dienten ebenfalls zum Datenschutz, da der Zeitstempel der pro gespeicherten Auftrag ebenfalls gespeichert wurde und somit genau zeigt, wann an diesen Beitrag gearbeitet wurde.

„findeAlleMitarbeiterimExternenSystem()“ wurde ebenfalls gebraucht für die Methode „sucheMitarbeiter()“. „sucheMitarbeiter()“ wurde erstellt um für die Methode „gesuchterMitarbeiter()“ die Daten der Mitarbeiter zu finden und zu entscheiden, ob der Mitarbeiter existiert oder es kein Eintrag existiert.

Es bestehen 3 Repositorien für Mitarbeiter, Rollen und Säulen je Tabelle in der Datenbank, die jeweils auch eine Interface-Klasse sind, da es die Schnittstelle zwischen Datenbank und Backend ist. Um die Details der Datenbankanbindung musste man sich nicht extra kümmern dank JPA. Diese Daten schickt die „MitarbeiterService“ danach auch wieder zurück zum Controller und wird somit für den Administrator angezeigt.

4.2.3 Erstellung der Controller-Klasse MitarbeiterController

Es wurde eine Controller-Klasse „MitarbeiterController“ für die Methoden zur Anwendungssteuerung erstellt. Die Klasse kümmert sich um das Auslesen und Umwandeln der JSON-Daten. Die Methode „mitarbeiterListe()“ liest das JSON-Array aus und wandelt es in Objekte einer Liste um und gibt diese an die Methode zurück, die alle Mitarbeiter-Daten in einer Tabelle anzeigen.

Die Methoden „rollenListe()“ und „saeulenListe()“ machen fast das Gleiche mit den Unterschied, dass diese die ausgelesenen JSON-Dateien in ein leeres Array packen, welche dann den Dropdown Rolle und Säule auf dem noch zu erstellenden Formular in Frontend füllen und die umgewandelten JSON Objekten dann auf dem Bildschirm wiedergeben. Die gesamte Controller-Klasse wurde mit den REST-Schnittstellen Annotation ausgestattet. REST-Schnittstellen bieten umfangreiche Unterstützung für die Erstellung von RESTful-Schnittstellen. Hierbei wurden Annotationen wie @RestController und @RequestMapping verwendet, um die Schnittstelle zu definieren.

Alle Listen-Methoden wurden mit einer „@GetMapping“ Annotation markiert, um die API je nach Namen der Methode zu definieren. Zum Beispiel die Methode „mitarbeiterListe()“ hat die Annotation „@GetMapping(„/mitarbeiterliste)“ als definierte Schnittstelle und REST-Endpunkt. Die Methoden „rollenListe()“ und „saeulenListe()“ sind mit der „/rollenliste“ und „/saeulenliste“ annotiert.

Um den Administrator es auch zu ermöglichen, die Rolle eines Mitarbeiters manuell zu verändern, brauchte es ebenfals eine Method namens „addMitarbeiter“. Die Annotation "@RequestParam" wurde verwendet, um Anfrageparameter in einer URL zu extrahieren und sie als Parameter in einer Methode zu benutzen. Die URL, die hier gemeint sind, sind die Endpunkte einer jeweiligen API (Zum Beispiel beinhaltet die Methode „mitarbeiterListe() die URL „http://localhost:3000/api/mitarbeiterliste“). Das Gleiche galt für die POST-Methode „addMitarbeiter“, wessen Endpunkt „/save“ ist. Um einen Mitarbeiter zu suchen und zu finden, wurde die Methode „gesuchterMitarbeiter()“ erstellt. Die Methode checkt ob es den Mitarbeiter gibt und sucht ihn per Email aus der Datenbank (in meinen Fall die PersonServiceMock.json Datei) und gibt diese dann aus. Falls diese Person nicht existiert, wird „Kein Mitarbeiter gefunden“ ausgegeben, wenn doch wird der Mitarbeiter mitsamt Email ausgegeben. Der Administrator verwaltet den Datenschutz und ist auch der einzige Mitarbeiter der Zugriff auf dieses Programm hat. Diese Methode arbeitet eng zusammen mit der „sucherMitarbeiter()“-Methode auf „MitarbeiterService“.

Login wurde nicht verwendet, da zu diesen Programm nur die Administoren Zugriff haben. Sich extra einzuloggen war daher eher aufwendig und unnötig zu gestalten.

4.2.4 Erstellung der DTO-Klassen

DTO-Klassen oder auch Datentransferobjekte sind Klassen, die ihren Gebrauch in CRUD-Programmen finden und gehören zu den Model-Klassen in der MVC Architektur. Die Datentransferobjekt Klassen definieren wie Daten über eine Applikation oder ein Netzwerk gesendet werden soll.

Die DTO-Klassen dieses Projekts sind „MitarbeiterResponse“, „RollenRequest“ und „SaeulenRequest“. Generell benutzte man die DTO-Klassen genutzt um die Daten von Server zu Frontend und andersrum zu transferieren, wie es der Name schon sagt. In dem Fall des Projekts sendete man die Daten aus der Model-Klasse Mitarbeiter, Rollen und Saeulen in eine

einzige Objekt-Klasse (MitarbeiterResponse) und übertrugen die Daten dann ans Frontend und andersrum (zum Beispiel von Frontend zu Backend, wenn man die Daten speichern wollte).

4.3 Anwendungsentwicklung mit JavaScript

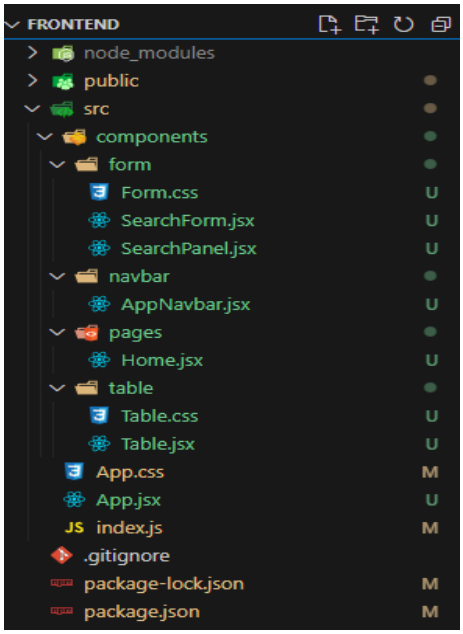
4.3.1 Erstellung des Frontend Projektordner

Bei der Erstellung des Frontend Projektordners wurde sich auf der IDE Visual Studio Code für das Framework Node.js entschieden. Dazu implementierte man die JavaScript-Bibliothek von React. Die Entscheidung fiel auf die React-Bibliothek, da man sich zeiteffizient und leicht eine minimalistische Oberfläche entwickeln wollte und man die meiste Erfahrung damit hatte. Da Node.js von Nöten war, um React zu implementieren, fiel die Entscheidung fürs Framework auch nicht schwer. Zum Implementieren von React wurde ebenfalls der Node Package Manager (kurz: npm) verwendet. Die Begründung lief darauf aus, dass Node Package Manager der Standard-Paketmanager von Node.js ist. Um einen Ordner zu erstellen, benutzte man das Terminal (Kommandozeile) in Visual Studio Code und tippte folgendes ein:

```
npx create-react-app abschlussprojekt
cd abschlussprojekt
npm start
```

Mit „npx create-react-app“ wurde der Ordner erstellt durch das Terminal. Nachdem man den vorherigen genannten Code in die Zeile eingibt, dauert es ein paar Minuten, bis alle Module und Bibliotheken in das Projekt implementiert sind. Mit „cd abschlussprojekt“ wurde in das Verzeichnis von dem gerade erstellten Projektordner gesteuert und mit „npm start“ wurde das Programm dann das erste mal ausgeführt. Wenn alles richtig lief, öffnete sich das gewählte Browser mit der vorgebenen Benutzeroberflächen-Schablone, die React als Platzhalter vorerst stellt, bis man das geändert hatte. Die installierten Packages und Module findet man in der Datei „package.json“. Man kann diese Datei ebenfalls manuell oder über Package Manager installieren. Bei React handelt es sich um Hierachie aus Komponenten, statt Klassen. Die App-Komponente gilt dabei als zentraler Einstiegspunkt für die React-Anwendung. Man nennt sie ebenfalls Wurzel-Komponente (root component) und man kann sie als „Main-Klasse“ React verstehen. Dort werden ebenfalls alle vorhandenen Komponenten zusammen gesetzt und ausgeführt.

Die Komponente „AppNavBar.jsx“ wurde für eine simple NavBar oberhalb der Seite eingerichtet, um es einfacher zu machen von einer Seite zur anderen Seite per Link zu gehen. Es existiert eine „Home.jsx“ als Homepage und es befindet sich ein Link auf ihr, welche direkt zum eigentlichen Programm führt. Es wird noch entschieden, ob man diese Seite behalten sollte, da Home.jsx sich eigentlich nur lohnt, wenn man mehrere Programme auf einer Seite zusammenfasst.



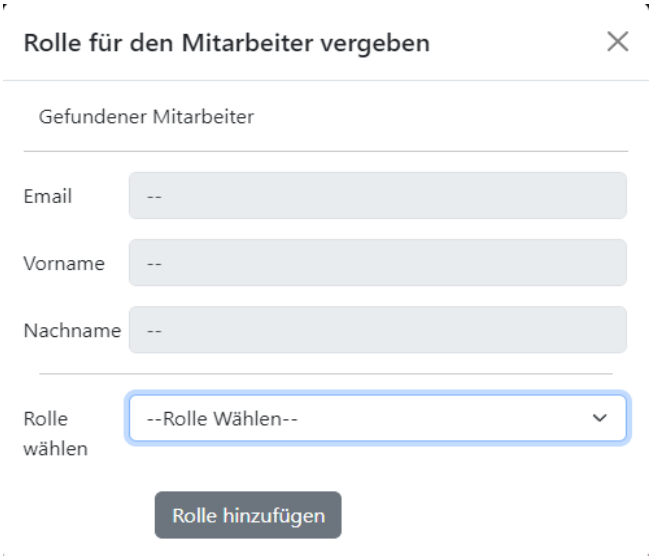
Projekt Hierarchie Frontend

4.3.2 Erstellung des Formulars

4.3.2.1 SearchPanel.jsx

„SearchPanel.jsx“ wurde geschrieben, um modale Dialoge zu erstellen. React Modal ist eine eingebaute Implementierung von React, die es ermöglicht modale Dialoge oder Popups-Fenster für React-Anwendungen zu erstellen. Modale Dialoge sind Overlay-Fenster, die über „SearchForm.jsx“, also den Hauptinhalt anzeigt werden und den Administrator auffordern eine Aktion auszuführen oder Informationen einzugeben. Dieses Fenster wird aufgerufen, wenn man auf den „SucheMitarbeiter“-Button klickt.

Es wird den Administrator auffordern, die Mitarbeiter-Daten, sofern vorhanden, manuell mit einer Rolle zu aktualisieren. Am Anfang wurden die Daten von der URL „http://127.0.0.1:9292/api/rollenliste“ also dem Backend und der Datenbank, die das Dialog-Fenster aus dem Backend bekam. Falls der Mitarbeiter nicht gefunden wurde, wurde „- -„ als Platzhalter hingestellt. Wenn man alles erledigt hatte und die Mitarbeiter-Daten vorhanden waren, beendete sich das Fenster von selbst nachdem man den Button fürs Absenden klickte und navigiert den Administator zurück zur Form-Page „/suche“ (SearchForm.jsx). Die Funktion „saveMitarbeiter“ im SearchPanel.jsx sendet die zu speichernden Daten dann zum Backend und protokolliert sie in der Datenbank. Die Daten werden nur lokal gespeichert und sind zum Protokollieren gedacht.



SearchPanel.jsx (Mit den Platzhalter „- -„)

4.3.2.2 SearchForm.jsx

„SearchForm.jsx“ ist eine Komponente wie „Table.jsx“, jedoch ist sie dafür zuständig, die Daten des Administrators zu verarbeiten und nicht nur anzuzeigen. „SearchForm.jsx“ beinhaltet die Felder „email“, „vorname“ und „nachname“ für von der useState()-Funktion. Die „useState“-Funktion ermöglicht es den Zustand in funktionalen Komponenten zu verwalten. Das Formular erwartet die Daten und Rows der Datenbank. Die Rollenliste und die gesuchten Mitarbeiter jedoch zeigen nur die 3 oben genannten Felder. Das Formular gibt ebenfalls die Backend Daten an die Tabebelle „Table.jsx“ weiter um ihre Felder zu füllen. „SearchForm.jsx“ und die Service-Klasse sind sich daher sehr ähnlich, da sie die Logik des Programms enthalten. Die Modal-Bibliothek wird auch über die „SearchForm.jsx“ verwaltet und eingestellt. Dazu hatten wir noch eine Funktion „getCurrentDate“ für die Zeitstempel entwickelt. Das war wichtig für die Protokollierung in der Datenbank und Ausgabe in der Tabelle auf dem Formular um den genauen Tag, Monat und Jahr anzuzeigen. Außerdem ist da noch ein „Prod-Säulen“ Dropdown, der jedoch auch eher für zukünftige Funktionen erstellt wurde.

SearchForm.jsx

4.3.3. Erstellung der Komponente Table.jsx

„Table.jsx“ bildet das Grundgerüst des Formulars. Mit der Funktion „getData()“ sucht er im Backend nach den abgesendeten Mitarbeiter-Daten. Dabei sucht er nach dem Endpunkt „api/mitarbeiterliste“, welche mit der Datenbank verbunden ist und fragt die Daten an. Danach gibt er diese Daten in einer implementierten HTML-Tabelle, die ebenfalls erstellt wurde.

Die Daten wurden dann in ein Array gespeichert, welches dann mit einer „.map“-Funktion ausgegeben wurde. Die Tabelle sollten die Daten, die nach der Eingabe des Administrators in SearchForm.jsx nach einem Neuladen der Seite angezeigt werden. In React oder generell JavaScript möchte man nicht dass die ganze Seite neugeladen wird. Die Begründung liegt darauf, das es einer besseren Benutzerfreundlichkeit dienen soll. Es wurde ebenfalls ein „Delete“-Button eingeführt, jedoch der befindet sich noch in Entwicklung. Damit wurde man leider nicht fertig, weil es zwar zum CRUD gehört, jedoch nicht zum Programm und seines Soll-Zustand. Es war kein Teil der Aufgabe.

Hier werden alle bearbeitete Mitarbeiter protokolliert

ID	Vorname	Nachname	Email	Datum	Verwalten
52	Trevar	Francisco	tfrancisco0@smh.com.au	2023-12-12	

Table.jsx mit vorhandenen Mitarbeiter-Daten

5. Qualitätssicherung

Zur Qualitätssicherung wurde sich für ein „Black-Box“ Test entschieden. Eine der Anforderungen wurde getestet, ob die Daten aus der Mock-Datei „PersonalServiceMock.json“ auf der Tabelle angezeigt werden. Nach 3 Test mit je 3 verschiedenen Mitarbeiter-Daten wurde es als fehlerfrei eingestuft. Es gab nur ein Problem: Man hatte die Rollen ausgelassen, die in der Tabelle einbezogen wurde, da es ein Problem gab diese Rollen auszugeben. Man kann die Rollen Richtigkeit daher nur in der Eingabe-Konsole in IntelliJ sehen und in der Datenbank. Zwischen Entwicklungssäule und Produktivsäule musste man nicht wechseln, dann man ja nur die Rolle für die Produktivsäule manuell nachgetragen hat.

PersonServiceMock.json

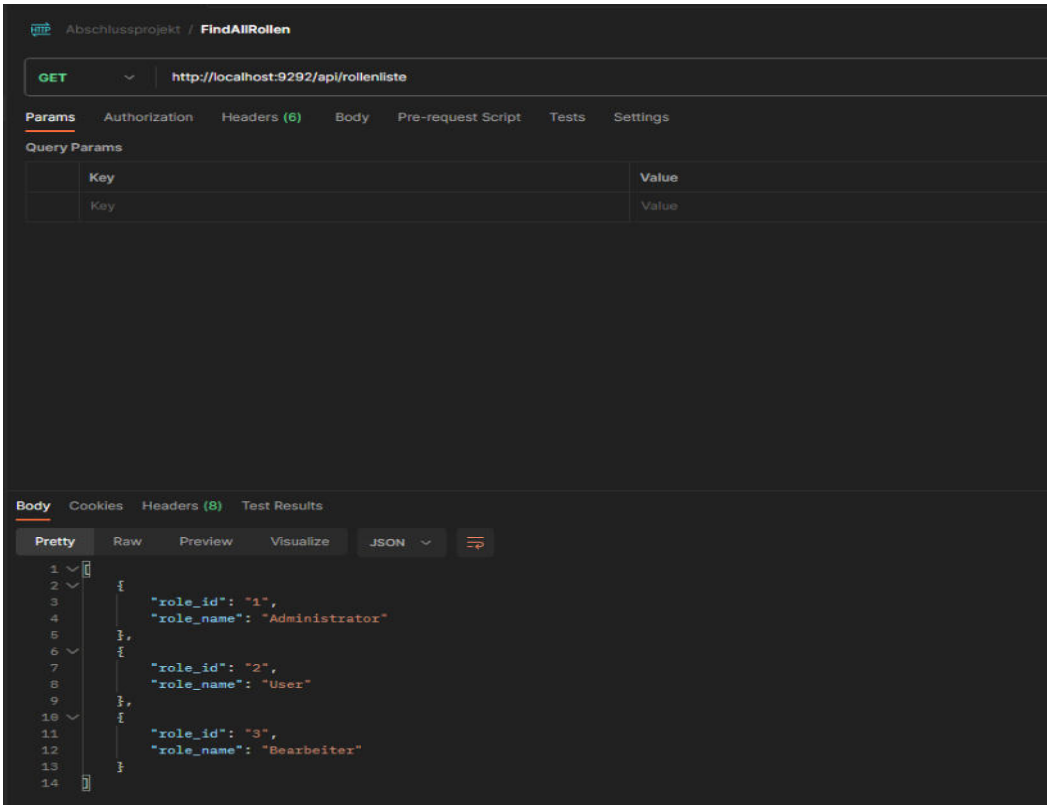
```
{"vorname": "Trevar", "nachname": "Francisco", "email": "tfrancisco0@smh.com.au"},
{"vorname": "Min", "nachname": "McQuilkin", "email": "mmcquilkin1@ycombinator.com"},
{"vorname": "Lesli", "nachname": "O' Timony", "email": "lotimony2@google.com"}
```

ID	Vorname	Nachname	Email	Datum	Verwalten
52	Trevar	Francisco	tfrancisco0@smh.com.au	2023-12-12	
53	Min	McQuilkin	mmcquilkin1@ycombinator.com	2023-12-12	
54	Lesli	O' Timony	lotimony2@google.com	2023-12-12	

3 Testversuche durch die Mock-Datei

Zum Testen der Schnittstellen hat man sich für das Programm „Postman“ entschieden. Postman wurde benutzt, bevor dass Frontend entwickelt war und half mit dem Testen der API. Für spätere Tests wurde dann das Frontend benutz. Postman ermöglicht es Entwicklern, HTTP-Anfragen an eine API zu senden und die Antworten zu überprüfen. Dabei können Tests definiert werden, um sicherzustellen, dass die API die erwarteten Ergebnisse liefert. Der erste Test wurde mit „http://localhost:9292/api/mitarbeiterliste“ und einer GET-Anfrage von Postman an die API getestet. Die Antwort zeigte alle 3 der oben im Bild angezeigten Mitarbeiter an. Damit war der Test bestanden.

Der zweite Test und letzte Test bevor man das Frontend zum Testen nutzte, war FindAllRollen, welche ebenfalls eine GET-Methode war. Diese verlief ebenfalls fehlerfrei. Somit war klar das die APIs von Backend einwandfrei funktionierten.



Test #2 zum Testen der API, ob sie alle Rollen findet und Datenbank Verbindung besteht

6. Fazit

Soll/Ist-Vergleich fand nur kurz statt, da am Ende alles erreicht wurde, was der Aufgabe und das Projekt von uns verlangte. Rollen wurden aktualisiert oder manuell zugefügt und keine Mitarbeiter Profile fehlten mehr Rollen auf dem Produktivserver. Der Abteilungsleiter war glücklich so waren wir es auch.

Das Projekt war eine ziemliche Herausforderung. Es war zwar nur ein kleines Tool zu einem viel großen Programm, jedoch hatte es viel Zeit und Nerven gekostet. Ich habe viel über Java, JavaScript, React, Spring Boot und allgemein CRUD-Anwendungen gelernt. Der Ablauf des Projekts hätte durchdachter und professioneller laufen können. Zum Abschlusssemester war es jedoch ziemlich viel, vor allem da noch andere Sachen wie die AP1 und AP2 während der Entwicklung des Projektes überstanden werden mussten. Man hatte mehr Zeit in die Entwicklung stecken können, hätte man mehr Zeit gehabt nach der AP2 noch das Projekt weiter zu machen. Lange Pausen waren in der Entwicklungszeit vorhanden, wo das Projekt wieder von neuen gelernt werden musste, weil nicht mehr klar war, was vor 3-4 Wochen man noch programmiert hat bzw. angefangen hat zu Programmieren. Die ausführliche wirtschaftliche Analyse wurde ausgelassen, so wie die Wirtschaftlichkeit in 3. Absatz. Es sind keine Kosten aufgekommen und deshalb war es unsinnig da trotzdem eine Wirtschaftlichkeit rauszuholen, denn das Programm ist für das TUHH selbst. Außerdem wurden auch keine Kosten ausgegeben bei den Lizenzen und bei den Arbeitsstunden, da ich von der TUHH nicht bezahlt wurde. Daher machte eine Amortisations-Rechnung keinen Sinn.

Der Zukunftsausblick auf die Rollenzuordnung ist das Programm zu erweitern mit einer Delete-Funktion als auch das Rollen Problem in der Table.jsx zu beheben. Ansonsten würde das Projekt als abgeschlossen gelten, da der Hauptgrund für dieses Tool weitestgehend erfüllt ist. Alle anderen übrig gebliebenen Empfehlungen werden dann wohl eher Luxus oder Benutzeroberflächen bedingt sein.

7. Anhang

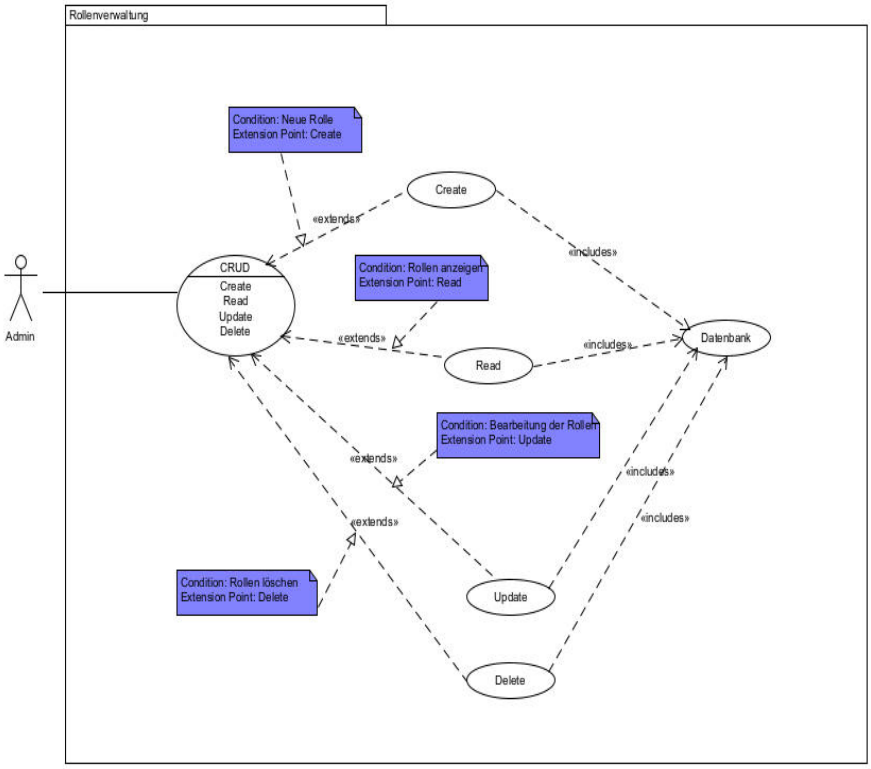
7.1 Glossar

Begriff	Erklärung
Framework	Ein Software Framework bietet eine Sammlung von Bibliotheken, Werkzeugen und Funktionen, die Entwicklern helfen, Anwendungen schneller und effizienter zu erstellen.
IDE	Eine integrierte Entwicklungsumgebung oder IDE stellt Programmierern eine Sammlung der wichtigsten Werkzeuge zur Softwareentwicklung unter einer Oberfläche zur Verfügung. Die Arbeit für die Erstellung von Programmen wird dadurch vereinfacht.
IntelliJ	IDE von Jet Brains
Visual Studio Code	IDE von Microsoft
Lombok	Lombok automatisiert die Erzeugung von Code und reduziert den Boilerplate-Code, den Entwickler normalerweise schreiben müssen. Außerdem generiert Lombok Getter, Setter und Konstruktoren, wodurch der Entwickler Zeit spart und der Code wird lesbarer und wartungsfreundlicher.
Node.js	Node.js ist eine serverseitige JavaScript-Laufzeitumgebung
Spring Boot	Siehe Framework
JPA	Java Persistence API (JPA) ist eine Spezifikation, die die Verwaltung von relationalen Daten in Java-Anwendungen beschreibt.
Spring Initializr	Der Initializr ist ein Tool, das es Entwicklern ermöglicht, schnell und einfach Spring Boot-Projekte zu erstellen.
React	React ist eine JavaScript-Bibliothek zur Entwicklung von Benutzeroberflächen
Maven	Maven ist ein leistungsstarkes Build-Management-Tool
pgAdmin4	PG Admin 4 ist eine Software, die entwickelt wurde, um die Verwaltung von PostGreSQL-Datenbanken zu erleichtern. Hier ist eine Erklärung zu den Funktionen und Eigenschaften von PG Admin 4

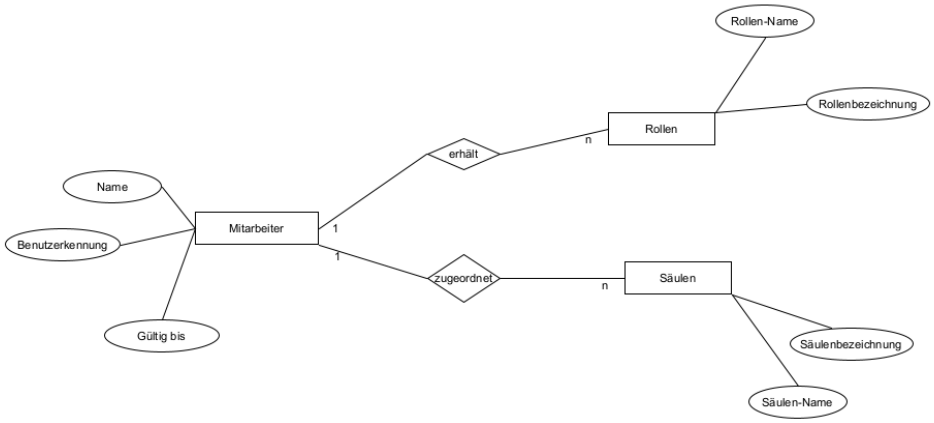
Reactstrap	React-Bibliothek die ermöglicht, Bootstrap-Komponenten in ihren React-Anwendungen zu verwenden.
Bootstrap	Kostenloses und Open-Source-CSS-Framework
Service Klasse	Die Service-Klasse in Spring enthält die gesamte Logik der Anwendung
Repository	Komponente um den Zugriff auf Datenbank und andere Datenquellen zu abstrahieren
Open Source	Eine Software, deren Quellcode öffentlich zugänglich ist und von der Gemeinschaft weiterentwickelt und verbessert werden kann.
Backend	Der Teil einer Software, der sich um die Datenverarbeitung, Datenbankanbindung und Geschäftslogik kümmert.
Frontend	Der Teil einer Software, der für die Benutzerschnittstelle und die Interaktion mit dem Benutzer verantwortlich ist.
XML	Extensible Markup Language - Eine Auszeichnungssprache zur Darstellung strukturierter Daten, die von verschiedenen Systemen gelesen und verarbeitet werden kann.
REST	"Representational State Transfer"-Softwarearchitekturstil, der entwickelt wurde, um die Gestaltung und Entwicklung von Webanwendungen zu leiten.
JAR/WAR	JAR (Java Archive) und WAR(Web Application Archive)
JSX	eine JavaScript-Erweiterung, die es ermöglicht, DOM-Bäume mithilfe einer XML-ähnlichen Syntax zu erstellen
Dependency Injection (DI)	eine Programmierungstechnik in der Softwareentwicklung. Dabei erhält ein Objekt oder eine Funktion andere Objekte oder Funktionen, die es benötigt, anstatt sie intern zu erstellen.
@Entity-Annotation	Signalisiert der Spring-Anwendung, dass es sich um eine Datenbank-Entität handelt.
@Controller-Annotation	Signalisiert der Spring-Anwendung, dass es sich um eine Controller-Klasse handelt.
Controller	Eine Controller-Klasse in Spring dient dazu, HTTP-Anfragen entgegenzunehmen und darauf zu reagieren. Sie verarbeitet die Anfragen und gibt die entsprechenden Antworten zurück.
Main-Klasse	Diese Methode ist der Einstiegspunkt für die Ausführung eines Java-Programms.

NPM	Node Package Manager – Node.js's Paketmanager
Komponente/Components	Eine Komponente in React ist eine unabhängige und wiederverwendbare Codeeinheit, die dazu dient, die Benutzeroberfläche (UI) in isolierte Teile aufzuteilen.
Form	Formular Bezeichnung in Webdesign
JSON	JavaScript Object Notation. Es handelt sich um ein offenes Standarddateiformat und Dateninterchange-Format, das menschenlesbaren Text verwendet, um Datenobjekte in Form von Attribut-Wert-Paaren und Arrays zu speichern und zu übertragen.
DTO	Datentransferobjekt
Modal	Overlay-Fenster

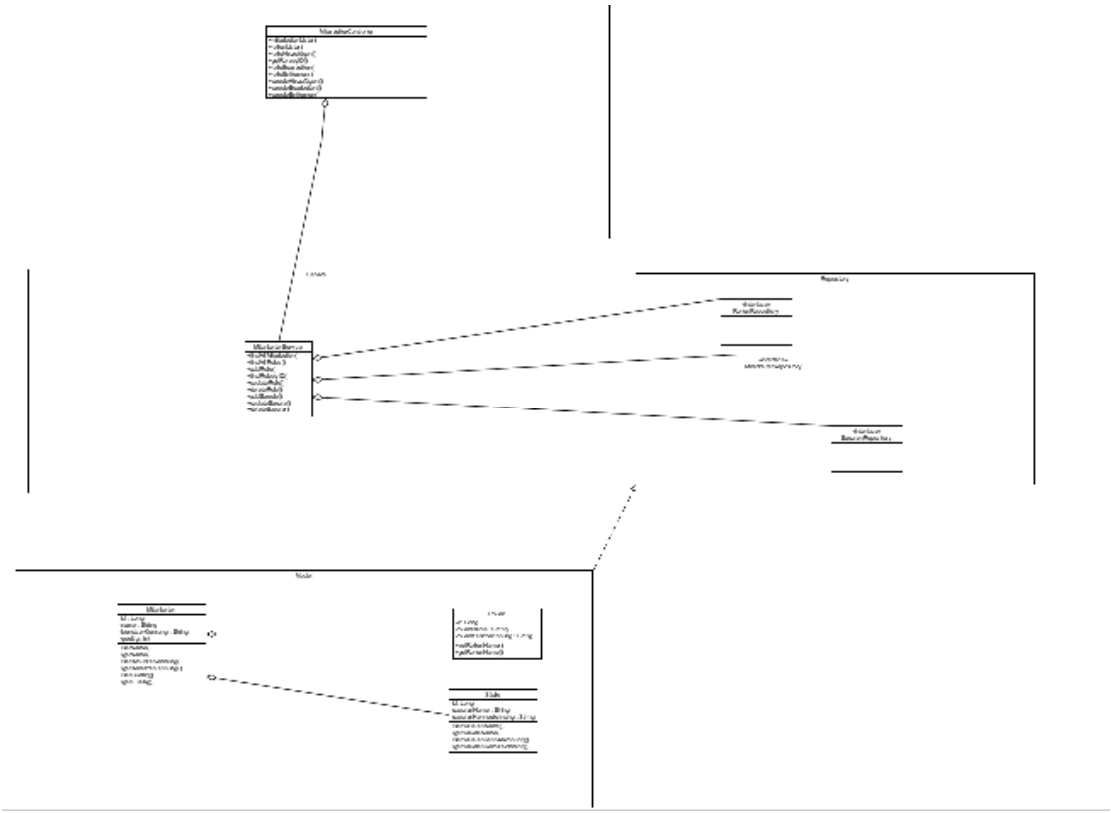
7.2 Diagramme



Anwendungsfalldiagramm



ERM-Diagramm



Klassendiagramm

7.3 Quellcode

```
package abschlussprojekt.demo.controller;

import abschlussprojekt.demo.dto.RollenRequest;
import abschlussprojekt.demo.dto.SaeulenRequest;
import abschlussprojekt.demo.model.Rollen;
import abschlussprojekt.demo.model.Saeule;
import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j2;
import abschlussprojekt.demo.model.Mitarbeiter;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import abschlussprojekt.demo.service.MitarbeiterService;

import java.net.URI;
import java.net.URISyntaxException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("api")
@AllArgsConstructor
@Log4j2
@CrossOrigin(origins = "http://localhost:9292")
public class MitarbeiterController {

    private final Logger logger =
LoggerFactory.getLogger(MitarbeiterController.class);

    @Autowired
    private MitarbeiterService mitarbeiterService;

    // Zeigt alle Mitarbeiter an.
    @GetMapping("/mitarbeiterliste")
    public ResponseEntity<List<Mitarbeiter>> mitarbeiterListe() {
        List<Mitarbeiter> mitarbeiterList =
mitarbeiterService.findAllMitarbeiter();
        return ResponseEntity.ok(mitarbeiterList);
    }

    /**
     * Speichere diese Mitarbeiter lokal
     * @param email
     * @param vorname
     * @param nachname
     * @return
     * @throws URISyntaxException
     */
    @PostMapping("/save")
    public ResponseEntity<Mitarbeiter>
addMitarbeiter(@RequestParam(value = "email", required = false)
String email,
```

```
@RequestParam(value = "vorname", required = false) String vorname,

@RequestParam(value = "nachname", required = false) String nachname
                                                        ) throws
URISyntaxException {

        Mitarbeiter m = new Mitarbeiter();
        m.setEmail(email);
        m.setVorname(vorname);
        m.setNachname(nachname);
        m.setTimestamp(LocalDate.now());
        m.setGueltig(1);
        log.info("Versuche die Rollen für einen Mitarbeiter
hinzufügen...");
        Mitarbeiter mit =
mitarbeiterService.saveRoleForMitarbeiter(m);
        return ResponseEntity.created(new
URI("/api/save")).body(mit);

    }

    @DeleteMapping("delete/{id}")
    public ResponseEntity<?> deleteMitarbeiter(@PathVariable Long
id){
        System.out.println("Mitarbeiter " + id + " wird geloscht");
        mitarbeiterService.deleteMitarbeiter(id);
        return ResponseEntity.ok().body("Erfolgreich gelöscht!");
    }

    @GetMapping("/searched-employee")
    public ResponseEntity<Mitarbeiter> gesuchterMitarbeiter(
            @RequestParam(value = "email", required = false) String
email,
            @RequestParam(value = "vorname", required = false) String
vorname,
            @RequestParam(value = "nachname", required = false)
String nachname
    ){

        System.out.println("Mitarbeiter mit " + email + " wird
gesucht...");

        Optional<Mitarbeiter> mitarbeiter =
Optional.ofNullable(mitarbeiterService.sucheMitarbeiter(email,
vorname, nachname));

        System.out.println(mitarbeiter.isEmpty() ?  "Keine
Mitarbeiter gefunden!"
                :  "Mitarbeiter mit " + email + " gefunden");


        return mitarbeiter.map(response ->
ResponseEntity.ok().body(response))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }


    /**
     * Holt sich die Rollen Liste aus der DB
```

```
     * @return
     */
    @GetMapping("/rollenliste")
    public ResponseEntity<List<RollenRequest>> rollenListe() {
        List<Rollen> rollenList = mitarbeiterService.findAllRoles();
        List<RollenRequest> rollenRequestList = new ArrayList<>();
        for (Rollen rollen: rollenList) {
            RollenRequest rollenRequest = new RollenRequest();
            rollenRequest.setRole_id(String.valueOf(rollen.getId()));
            rollenRequest.setRole_name(rollen.getRollenName());
            rollenRequestList.add(rollenRequest);
            System.out.println("Gefunden: " +
rollenRequest.getRole_name());
        }

        return ResponseEntity.ok(rollenRequestList);
    }

    /**
     * holt sich die Säulen Liste aus der DB
     * @return
     */
    @GetMapping("/saeulenliste")
    public ResponseEntity<List<SaeulenRequest>> sauelenListe() {
        List<Saeule> saeuleList =
mitarbeiterService.findAllSaeulen();
        List<SaeulenRequest> saeulenRequestList = new ArrayList<>();
        for (Saeule saeule : saeuleList) {
            SaeulenRequest saeulenRequest = new SaeulenRequest();
            saeulenRequest.setSaeule_id(saeule.getId());
            saeulenRequest.setSaeule_name(saeule.getSaeulenName());
            saeulenRequestList.add(saeulenRequest);
            System.out.println("Gefunden: " +
saeulenRequest.getSaeule_name());

        }
        return ResponseEntity.ok(saeulenRequestList);
    }
```

```
package abschlussprojekt.demo.service;

import abschlussprojekt.demo.model.Mitarbeiter;
import abschlussprojekt.demo.model.Rollen;
import abschlussprojekt.demo.model.Saeule;
import abschlussprojekt.demo.repository.SaeulenRepository;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.ClassPathResource;
import org.springframework.stereotype.Service;
import abschlussprojekt.demo.repository.MitarbeiterRepository;
import abschlussprojekt.demo.repository.RollenRepository;

import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;


@Service
public class MitarbeiterService {

    private final Logger logger =
LoggerFactory.getLogger(this.getClass());
    @Autowired
    private MitarbeiterRepository mitarbeiterRepository;

    @Autowired
    private RollenRepository rollenRepository;

    @Autowired
    private SaeulenRepository h;

    public List<Mitarbeiter> findAllMitarbeiter() {
        return mitarbeiterRepository.findAll();
    }

    /**
     * Konvertitert ein JSON File zu Model Object
     *
     * @param classType
     * @param <T>
     * @return
     */
    private List<Mitarbeiter> findeAlleMitarbeiterImExternenSystem()
{

        List<Mitarbeiter> mList = new ArrayList<>();

        try {
            File file = new ClassPathResource("data/"+
"PersonServiceMock.json").getFile();

            try (BufferedReader bufferedReader = new
BufferedReader(new FileReader(file))) {
                String line;
                while ((line = bufferedReader.readLine()) != null) {
                    ObjectMapper mapper = new ObjectMapper();

mapper.disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
                    Mitarbeiter mitarbeiter = mapper.readValue(line,
Mitarbeiter.class);

                    mList.add(mitarbeiter);
                    logger.debug( mitarbeiter.getEmail() + " in die
Liste hinzugefügt!");

                }
            }
            catch (Exception e) {
                System.out.println("Label Update fehlgeschlagen");
            }
```

```
        } catch (Exception e) {
            logger.error(e.getMessage());
        }

        return mList;
    }

    /**
     * Lösche Einträge aus der DB
     */
    public void deleteMitarbeiter(Long id){
        mitarbeiterRepository.deleteById(id);
    }

    /**
     * rolle hinzufügen und in die DB protokollieren
     * @param mitarbeiter
     * @return
     */
    public Mitarbeiter saveRoleForMitarbeiter(Mitarbeiter
mitarbeiter) {
        //füge die Rolle hinzu und speichert sie Lokal
        return mitarbeiterRepository.save(mitarbeiter);
    }



    /**
     * finde Mitarbeiter aus einem MockService Json File
     * @param email
     * @param vorname
     * @param nachname
     * @return
     */
    public Mitarbeiter sucheMitarbeiter(String email, String vorname,
String nachname){

        List<Mitarbeiter> mitarbeiterList =
findeAlleMitarbeiterImExternenSystem();

        System.out.println("Liste Größe: " + mitarbeiterList.size());

        Mitarbeiter h = new Mitarbeiter();

        for(Mitarbeiter mitarbeiter : mitarbeiterList){

                if(mitarbeiter.getNachname().equals(nachname)
                        && mitarbeiter.getVorname().equals(vorname)
                        && mitarbeiter.getEmail().equals(email)) {

                    searchedEntry.setEmail(mitarbeiter.getEmail());

searchedEntry.setVorname(mitarbeiter.getVorname());

searchedEntry.setNachname(mitarbeiter.getNachname());

                    System.out.println("Mitarbeiter gefunden: "
                            + searchedEntry.getEmail() + " "
                            + searchedEntry.getNachname() + " "
                            + searchedEntry.getVorname());
```

```
                }

            }

            return searchedEntry;
        }
        public List<Rollen> findAllRoles() {
            return rollenRepository.findAll();
        }

        public List<Saeule> findAllSaeulen() {
            return saeulenRepository.findAll();
        }
```

```
package abschlussprojekt.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDate;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Mitarbeiter {

    @Id
    @GeneratedValue
    private Long id;
    private String vorname;
    private String nachname;
    private String email;
    private int gueltig;
    private LocalDate timestamp;
}
package abschlussprojekt.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
```

```
public class Rollen {

    @Id
    @GeneratedValue
    private Long id;
    private String rollenName;
    private String rollenKennzeichnung;


    /*insert into rollen (id, rollen_kennzeichnung, rollen_name)
values (1, 'ADMIN', 'Administrator');
    insert into rollen (id, rollen_kennzeichnung, rollen_name) values
(2, 'USER', 'User');
    insert into rollen (id, rollen_kennzeichnung, rollen_name) values
(3, 'BEARB', 'Bearbeiter');*/
}
```

```
package abschlussprojekt.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Saeule {

    @Id
    @GeneratedValue
    private Long id;
    private String saeulenName;
    private String saeulenKennzeichnung;
}
package abschlussprojekt.demo.repository;

import abschlussprojekt.demo.model.Mitarbeiter;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

@Repository
public interface MitarbeiterRepository extends
JpaRepository<Mitarbeiter, Long> {

     @Query(value="SELECT * FROM mitarbeiter m WHERE " +
             "m.email = 'test1@test.de' " +
             "and m.vorname = 'test1' " +
             "and m.nachname= 'test1'", nativeQuery = true)
     Mitarbeiter getMitarbeiter(String email, String vorname, String
nachname);
}
```

```java
package abschlussprojekt.demo.repository;

import abschlussprojekt.demo.model.Mitarbeiter;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

@Repository
public interface MitarbeiterRepository extends
JpaRepository<Mitarbeiter, Long> {

     @Query(value="SELECT * FROM mitarbeiter m WHERE " +
             "m.email = 'test1@test.de' " +
             "and m.vorname = 'test1' " +
             "and m.nachname= 'test1'", nativeQuery = true)
     Mitarbeiter getMitarbeiter(String email, String vorname, String
nachname);
}
package abschlussprojekt.demo.repository;

import abschlussprojekt.demo.model.Rollen;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RollenRepository extends JpaRepository<Rollen, Long>
{}
```

```java
package abschlussprojekt.demo.repository;

import abschlussprojekt.demo.model.Saeule;
import org.springframework.data.jpa.repository.JpaRepository;

public interface SaeulenRepository extends JpaRepository<Saeule,
Long> {}
```

```java
package abschlussprojekt.demo.dto;

import lombok.Data;

import java.time.LocalDate;

@Data
public class MitarbeiterResponse {
    private String vorname;
    private String nachname;
    private String email;
    private String benutzerkennung;
    private String rollenName;
    private String saeulenName;
    private LocalDate timestamp;
}
```

```java
package abschlussprojekt.demo.dto;

import lombok.Data;

@Data
public class RollenRequest {
    private String role_id;
    private String role_name;
}
```

```java
package abschlussprojekt.demo.dto;


import lombok.Data;

@Data
public class SaeulenRequest {
    private Long saeule_id;
    private String saeule_name;
}
package abschlussprojekt.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AbschlussprojektApplication {

      public static void main(String[] args) {
            SpringApplication.run(AbschlussprojektApplication.class,
args);
      }
}
```

```
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.show-sql=true
spring.datasource.url=jdbc:postgresql://localhost:5432/tuhhprojekt
spring.datasource.username=postgres
spring.datasource.password=postgres

server.port=9292
```

```
{"vorname": "Trevar", "nachname": "Francisco", "email":
"tfrancisco0@smh.com.au"},
{"vorname": "Min", "nachname": "McQuilkin", "email":
"mmcquilkin1@ycombinator.com"},
{"vorname": "Lesli", "nachname": "O' Timony", "email":
"lotimony2@google.com"}
```

```
import React, { useState, useEffect } from 'react'
import {
    Label, Form, FormGroup, Input, Col,
    FormText, Button, Container, Row, Modal, ModalHeader, ModalBody, ModalFooter
} from "reactstrap"
import Table from '../table/Table'
import SearchPanel from './SearchPanel';

function SearchForm() {

    /**
     * form state data
     */
    const [form, setForm] = useState({
        vorname: '',
        nachname: '',
        email: '',
        saeulenName: '',
        timestamp: ''
    });
    //state für alle geladene Säulen
    const [saeulen, setSaeulen] = useState([])

    // gesuchter Mitarbeiter Daten
    const [gesuchterMitarbeiter, setGesuchterMiterbeiter] = useState({
        email: '',
        vorname: '',
        nachname: '',
    })
    //modal open / close state
    const [modal, setModal] = useState(false);
    /**
     *
     * @returns setze Modal Fenster auf oder zu
     */
    const toggle = () => {
        setModal(!modal)
    };
    /**
     * suche im backend nach diesem Mitarbeiter
     */
    const findSerchedMitarbeiter = (email, vorname, nachname) => {
        console.log("findSearchedMitarbeiter() aufruf")

        fetch(`api/searched-employee?email=${email}&vorname=${vorname}&nachname=${nachname}`,
            {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                return response.json();
            })
            .then(function (jsonResponse) {
                setGesuchterMiterbeiter(jsonResponse)
                console.log("findSearchedMitarbeiter State: " + JSON.stringify(gesuchterMitarbeiter))
            });

    }

    /**
    * am anfang sollen die daten geladen werden
    * http://127.0.0.1:9292/api/saeulenliste
    */
    const getSaeuleData = () => {
        fetch('api/saeulenliste',
            {
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                console.log(response)
                return response.json();
            })
            .then(function (jsonResponse) {
                console.log("Säulen Response: " + jsonResponse);
                setSaeulen(jsonResponse)
            });
    }
    /**
     * am anfang immer die Säulen laden
     */
    useEffect(() => {
        getSaeuleData()
    }, []);

    /**
     * schreibe ins State wenn Input geändert wird
     * @param {} event
     */
    const handleChange = (event) => {
        setForm({
            ...form,
            [event.target.id]: event.target.value,
        });
    };

    /**
     *
     * @returns gibt aktuelles Datum zurück
     */
    const getCurrentDate = () => {
        var today = new Date();
        var month = today.getMonth() + 1;
        var year = today.getFullYear();
        var day = today.getDate();
        var currDate = `${day}.${month}.${year}`;
        console.log("Timestamp: " + currDate);
        return currDate;
    }

    /**
     * hier posten data auf server endpoint
     * @param {*} event
     */
    const handleSubmit = (event) => {
        event.preventDefault()

        //aktueller Zeit
        form.timestamp = getCurrentDate

        //Daten anzeigen
        console.log("Suche nach: " + JSON.stringify(form))

        //reset Modal Form immer am Anfang
        setGesuchterMiterbeiter({
            email: "",
            vorname: "",
            nachname: ""
        })

        //API Call Suche ob in CampusManagement System dieser Mitarbeiter gefunden wird
        findSerchedMitarbeiter(form.email, form.vorname, form.nachname)

        //zusätz Modal Fenster
        toggle()


    };


                <Modal isOpen={modal} toggle={toggle}>
                    <ModalHeader toggle={toggle}>Rolle für den Mitarbeiter vergeben</ModalHeader>
                    <ModalBody>
                        <SearchPanel gesuchterMitarbeiter={gesuchterMitarbeiter} toggle={toggle} />
                    </ModalBody>
                </Modal>

                <Container>
                    <hr />
                    Hier werden alle bearbeitete Mitarbeiter protokolliert
                    <hr />
                </Container>
                <Container>
                    <Table />
                </Container>
            </Row>
        </Container >


    )
}

export default SearchForm
```

```
    return (
        <Container style={{ border: "1px solid #ccc", paddingTop: "3rem", paddingBottom: "3rem" }}>
            <Col>
                <h3>
                    TUHH-Rollenzuordnung
                </h3>
                <Row style={{ marginBottom: "2rem" }}>
                    <Container>
                        <hr />
                        <h5>Suche</h5>
                    </Container>

                </Row>

            </Col>
            <Row >
                <Col xs="10">
                    <Form onSubmit={handleSubmit}>
                        <FormGroup row>
                            <Label for="email" sm={2}>Email </Label>
                            <Col sm={10}>
                                <Input
                                    id="email"
                                    name="email"
                                    placeholder="Email"
                                    type="email"
                                    value={form.email}
                                    onChange={handleChange}
                                    required
                                />
                            </Col>
                        </FormGroup>
                        <FormGroup row>
                            <Label for="vorname" sm={2}>Vorname</Label>
                            <Col sm={10}>
                                <Input
                                    id="vorname"
                                    name="vorname"
                                    placeholder="Vorname"
                                    type="text"
                                    value={form.vorname}
                                    onChange={handleChange}
                                    required
                                />
                            </Col>
                        </FormGroup>
                        <FormGroup row>
                            <Label for="nachname" sm={2}>Nachname</Label>
                            <Col sm={10}>
                                <Input
                                    id="nachname"
                                    name="nachname"
                                    placeholder="Nachname"
                                    type="text"
                                    value={form.nachname}
                                    onChange={handleChange}
                                    required
                                />
                            </Col>
                        </FormGroup>
                        <FormGroup row>
                            <Label for="saeuleName" sm={2}></Label>

                            <Col sm={10}>
                                <Input
                                    disabled
                                    id="saeuleName"
                                    name="saeuleName"
                                    type="select"
                                    onChange={handleChange}
                                >
                                    <option>Prod Säule</option>
                                    {
                                        saeulen.map((saeule) => (
                                            <option key={saeule.saeule_id} value={saeule.saeule_name}>
                                                {saeule.saeule_name}
                                            </option>

                                        ))
                                    }

                                </Input>


                            </Col>


                        </FormGroup>
                        <FormGroup >
                            <Col sm={{ offset: 10, size: 30 }} >
                                <Button color="danger">
                                    Suche Mitarbeiter
                                </Button>
                            </Col>
                        </FormGroup>
                    </Form>
                </Col>



                <Modal isOpen={modal} toggle={toggle}>
                    <ModalHeader toggle={toggle}>Rolle für den Mitarbeiter vergeben</ModalHeader>
                    <ModalBody>
                        <SearchPanel gesuchterMitarbeiter={gesuchterMitarbeiter} toggle={toggle} />
                    </ModalBody>
                </Modal>


                <Container>
                    <hr />
                    Hier werden alle bearbeitete Mitarbeiter protokolliert
                    <hr />
                </Container>
                <Container>
                    <Table />
                </Container>
            </Row>
        </Container >


    )
}

export default SearchForm
```

```
    import React, { useEffect, useState } from 'react'
import "./Table.css"
import { RiDeleteBin2Line } from "react-icons/ri";

function Table() {


    const [data, setData] = useState([])


    /**
         * get mitarbeiter daten
         */
    const deleteData = (id) => {
        fetch(`api/delete/${id}`,
            {
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                console.log(response)
                return response.json()
            })
            .then(function (jsonResponse) {
                console.log(jsonResponse)
                setData(jsonResponse)
            });
    }


    /**
     * get mitarbeiter daten
     */
    const getData = () => {
        fetch('api/mitarbeiterliste',
            {
                method: "GET",
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                console.log(response)
                return response.json()
            })
            .then(function (jsonResponse) {
                console.log(jsonResponse)
                setData(jsonResponse)
            });
    }
    /**
     * set data am anfang in state
     */
    useEffect(() => {
        getData()
    }, [])

    return (

        <table className='table'>
            <thead>
                <tr>
                    <th className='header__item'>
                        <span>ID</span>
                    </th>
                    <th className='header__item'>
                        <span>Vorname</span>
                    </th>
                    <th className='header__item'>
                        <span>Nachname</span>
                    </th>
                    <th className='header__item'>
                        <span>Email </span>
                    </th>
                    <th className='header__item'>
                        <span>Datum </span>
                    </th>
                    <th className='header__item'>
                        <span>Verwalten</span>
                    </th>

                </tr>
            </thead>
            <tbody>


                {data.map((obj, index) => (
                    <tr key={index}>
                        <td>{obj.id}</td>
                        <td>{obj.vorname}</td>
                        <td>{obj.nachname}</td>
                        <td>{obj.email}</td>
                        <td>{obj.timestamp}</td>
                        <td>
                            <a onClick={() => deleteData(obj.id)}>
                                <RiDeleteBin2Line />
                            </a>

                        </td>
                    </tr>
                ))}


            </tbody>
        </table >

    )
}

export default Table;
```

```
import React, { useState, useEffect } from 'react'
import { useNavigate } from 'react-router-dom'
import {
    Label, Form, FormGroup, Input, Col,
    FormText, Button, Container, Row, Modal, ModalHeader, ModalBody, ModalFooter
} from "reactstrap"


function SearchPanel({ gesuchterMitarbeiter, toggle }) {

    /**
     * zum navigieren zu andere Seiten
     */
    const navigate = useNavigate();
    /**
    * form state data
    */
    const [form, setForm] = useState({
        rollenName: '',
    });
    //state für alle geladene Rollen
    const [roles, setRoles] = useState([])

    const [disabledInput, setDisabledInput] = useState(true)

    const setSaveButton = () => {
        if (gesuchterMitarbeiter.email != '') {
            setDisabledInput(false)
        }
    }


    /**
      * am anfang sollen die daten geladen werden
      * http://127.0.0.1:9292/api/rollenliste
      */
    const getRolesData = () => {
        fetch('api/rollenliste',
            {
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                console.log(response)
                return response.json();
            })
            .then(function (jsonResponse) {
                console.log("Rollen Response: " + jsonResponse);
                setRoles(jsonResponse)
            });
    }
    /**
    * am anfang immer laden
    */
    useEffect(() => {
        getRolesData()
    }, []);

    /**
     * setze Daten wenn feld geändert wird
     * @param {} event
     */
    const handleChange = (event) => {
        setForm({
            ...form,
            [event.target.id]: event.target.value,
        });
    };

    /**
     * hier posten data auf server endpoint
     * @param {*} event
     */
    const saveData = (event) => {
        //verhindere Browser refresh
        event.preventDefault()

        //Zeige alle Form Daten in die Konsole
        console.log("Rolle: " + form.rollenName)
        console.log("Email: " + gesuchterMitarbeiter.email)
        console.log("Vorname: " + gesuchterMitarbeiter.vorname)
        console.log("Nachname: " + gesuchterMitarbeiter.nachname)

        //Speicher in die DB
        saveMitarbeiter(gesuchterMitarbeiter.email,
            gesuchterMitarbeiter.vorname,
            gesuchterMitarbeiter.nachname,
            form.rollenName)
        //schalte Modal aus
        toggle()

        navigate(`/suche`)
    };


    /**
    * suche im backend nach diesem Mitarbeiter
    */
    const saveMitarbeiter = (email, vorname, nachname, roleName) => {
        console.log("findSearchedMitarbeiter() aufruf")

        fetch(`api/save?email=${email}&vorname=${vorname}&nachname=${nachname}`,
            {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Accept': 'application/json'
                }
            }
        )
            .then(function (response) {
                return response.json();
            })
            .then(function (jsonResponse) {
                console.log("save Mitarbeiter json response: " + jsonResponse)
            });
    }
```

```
    return (
        <>

            <Container>
                Gefundener Mitarbeiter
            </Container>
            <hr />
            <Form onSubmit={saveData}>
                <FormGroup row>
                    <Label for="email" sm={2}>Email</Label>
                    <Col sm={10}>
                        <Input
                            disabled
                            id="email"
                            name="email"
                            placeholder="--"
                            type="email"
                            value={gesuchterMitarbeiter.email}
                        />
                    </Col>
                </FormGroup>
                <FormGroup row>
                    <Label for="vorname" sm={2}>Vorname</Label>
                    <Col sm={10}>
                        <Input
                            disabled
                            id="vorname"
                            name="vorname"
                            placeholder="--"
                            type="text"
                            value={gesuchterMitarbeiter.vorname}

                        />
                    </Col>
                </FormGroup>
                <FormGroup row>
                    <Label for="nachname" sm={2}>Nachname</Label>
                    <Col sm={10}>
                        <Input
                            disabled
                            id="nachname"
                            name="nachname"
                            placeholder="--"
                            type="text"
                            value={gesuchterMitarbeiter.nachname}

                        />
                    </Col>
                </FormGroup>
                <Container>
                    <hr />
                </Container>
                <FormGroup row>
                    <Label for="rollenName" sm={2}> Rolle wählen</Label>

                    <Col sm={10}>
                        <Input
                            id="rollenName"
                            name="rollenName"
                            type="select"
                            onChange={handleChange}
                        >
                            <option>--Rolle Wählen--</option>
                            {

                                roles.map((role) => (
                                    <option key={role.role_id} value={role.role_name}>
                                        {role.role_name}
                                    </option>

                                ))
                            }

                        </Input>
                    </Col>
                </FormGroup>

                <FormGroup check row >
                    <Col sm={{ offset: 2, size: 10 }} >
                        <Button>
                            Rolle hinzufügen
                        </Button>
                    </Col>
                </FormGroup>
            </Form>




        </>
    )
}
export default SearchPanel
```

```jsx
import React, { useState } from 'react';
import { Collapse, Nav, Navbar, NavbarBrand, NavbarToggler, NavItem, NavLink } from 'reactstrap';
import { Link } from 'react-router-dom';

const AppNavbar = () => {

    const [isOpen, setIsOpen] = useState(false);

    return (
        <Navbar color='dark' dark expand="md">
            <NavbarBrand tag={Link} to="/">
                Home
            </NavbarBrand>
            <NavbarToggler onClick={() => {
                setIsOpen(!isOpen)
            }} />
            <Collapse isOpen={isOpen} navbar>

                <Nav className='justify-content-end' style={{ width: "100%" }} navbar>

                    <NavItem>
                        <NavLink href='/'> TUSP </NavLink>
                    </NavItem>

                </Nav>
            </Collapse>
        </Navbar>
    )
}

export default AppNavbar;
```

```jsx
import React from 'react';

import { Button, Container } from 'reactstrap';
import { Link } from 'react-router-dom';



const Home = () => {
    return (
        <div>
            <Container fluid>
                <Button color="link"><Link to="/suche">Suche Mitarbeiter im externen System und weise die
Rollen zu</Link></Button>
            </Container>
        </div>

    )
}

export default Home;
```

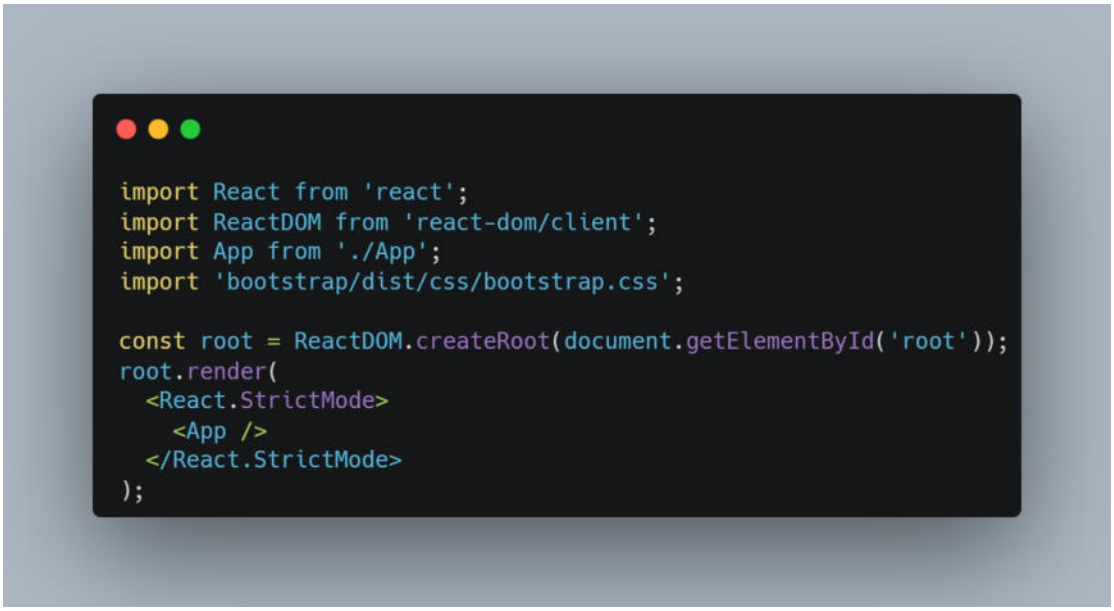
```
import './App.css';
import React from 'react';
import SearchForm from './components/form/SearchForm';
// import Table from './components/table/Table';
import Home from './components/pages/Home';
import AppNavbar from './components/navbar/AppNavbar';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

function App() {

  return (
    <Router>
      <AppNavbar />
      <Routes>
        <Route exact path='/' element={<Home />}></Route>
        <Route exact path='/suche' element={<SearchForm />}></Route>
      </Routes>
    </Router>
  )

}
export default App;
```



```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import 'bootstrap/dist/css/bootstrap.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

```
{
  "name": "mitarbeiter-rollen-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^5.3.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^4.12.0",
    "react-router-dom": "^6.20.1",
    "react-scripts": "5.0.1",
    "reactstrap": "^9.2.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "proxy": "http://localhost:9292",
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```


table {
border-collapse: collapse;
width: 100%;
}
th,
td {
padding: 8px;
text-align: left;
border-bottom: 1px solid #ddd;
}
tr:hover {
background-color: coral;
}<App />
</React.StrictMode>
);

```css
.form-container {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 100%;
    border: 2px solid red;
    overflow: hidden;

}

.form-field {
    display: flex;
    flex-direction: column;
    align-items: flex-start;
    width: 500px;
    padding-top: 20px;
}

.form-field-input {
    font-size: 1.3rem;
    width: 95%;
    height: 30px;
    /* margin: 10px auto; */
    padding: 10px 10px;
}

.form-button {
    margin: 20px auto;
    display: block;
    width: 500px;
    padding-top: 20px;
    padding-bottom: 20px;
    font-size: 1.3rem;
}

.select-dropdown {
    font-size: 1.3rem;
    width: 100%;
    height: 50px;
    /* margin: 10px auto; */
    padding: 10px 10px;

}
```

Die Platzhalter und Standard Klassen/Komponente wurden nicht im Quellcode berücksichtig, da sie keine Verwendung haben.