Technische Dokumentation

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Verantwortlichkeiten

Im Rahmen unseres Softwareentwicklungsprojekts an der Universität hat jedes Teammitglied spezifische Rollen und Verantwortlichkeiten übernommen, um eine strukturierte und effiziente Zusammenarbeit zu gewährleisten. Die folgende Auflistung gibt einen Überblick über die Hauptaufgaben und Zuständigkeiten der einzelnen Mitglieder sowie die Organisation unserer Arbeitsweise.

Rollen und Zuständigkeiten

Justus (Projektleiter):

Justus war der Projektleiter und verantwortete den Soll-Ist-Vergleich sowie das Management der Teamaufgaben. Er leitete die wöchentlichen Teammeetings, war für die Kommunikation mit dem Projektbetreuer zuständig und präsentierte die Ergebnisse des Teams. Seine Fähigkeit, Aufgaben zu delegieren und die Übersicht zu behalten, trug maßgeblich zur erfolgreichen Durchführung des Projekts bei.

Basti (Stellvertretender Projektleiter):

Als rechte Hand von Justus unterstützte Basti in allen Aspekten der Projektarbeit. Er half bei der Organisation, war an verschiedenen Aufgaben beteiligt und unterstützte vor allem bei der Durchführung von Tests sowie der allgemeinen Problembehebung.

Phillip (Backend-Verantwortlicher):

Phillip war hauptverantwortlich für die Backend-Entwicklung. Er überprüfte alle Commits, sorgte für die Einhaltung der Qualitätsstandards im Code und stand bei technischen Fragen rund um das Backend als Ansprechpartner zur Verfügung. Gemeinsam mit Jonas legte er das Fundament für die Backend-Architektur.

Jonas:

Jonas war für das Zeitmanagement zuständig und unterstützte in vielen Bereichen, insbesondere in der Anfangsphase, durch die Implementierung zentraler Funktionen. Zusammen mit Justus erstellte er wöchentliche Excel-Sheets mit zugeteilten Aufgaben. Er war auch maßgeblich an der Erstellung des Backend-Fundaments beteiligt und arbeitete teamübergreifend mit. Außerdem spielte er eine entscheidende Rolle im Frontend, wo er sämtliche von ihm entwickelte Funktionen abbildete.

Franz S.:

Franz unterstützte die Backend-Entwicklung und trug zur Implementierung spezifischer Features bei. Seine Zusammenarbeit mit Phillip und Exer half dabei, die Backend-Entwicklung effizient zu gestalten. Zudem fertigte er weitere Diagramme an die für die Planung und schließlich für die Dokumentation verwendet werden konnte.

Daniel R.:

Daniel R. half in verschiedenen Bereichen des Projekts mit und konzentrierte sich gegen Ende verstärkt auf die Erstellung des Frontends und Anpassungen für mobile Geräte. Sein Einsatz für die Qualitätssicherung der Benutzeroberfläche war ein wichtiger Beitrag zur Fertigstellung des Projekts.

Qipeng:

Qipeng war hauptverantwortlich für die Frontend-Entwicklung. Er implementierte wesentliche UI-Komponenten und sorgte dafür, dass das Frontend sowohl funktional als auch ästhetisch den Anforderungen entsprach.

Daniel G. (Datenbank-Verantwortlicher):

Daniel G. war für die Erstellung und Pflege des Datenmodells sowie die Einrichtung der Datenbank zuständig. Er stellte sicher, dass die Datenbank korrekt mit dem Backend verbunden war und half zudem bei der Implementierung von Funktionen. Zusätzlich führte er gemeinsam mit Basti Tests durch, um die Qualität und Funktionalität des Fremdsystems zu gewährleisten.

Amran (Präsentations-Verantwortlicher:

Amran übernahm die Verantwortung für die Erstellung und Präsentation der Projektergebnisse. Neben seiner Hauptaufgabe trug er durch die Entwicklung einiger Frontend-Seiten zur Gesamtarbeit bei.

Exer (Backendentwickler):

Exer war maßgeblich an der Implementierung des Backends beteiligt und unterstützte dabei die anderen Entwickler mit seiner Expertise in der Backend-Programmierung. Zudem brachte er diese expertise in die Projektdokumentation mit.

Zusammenarbeit und Arbeitsweise

- **Teammeetings:** Jeden Donnerstag um 18:00 Uhr fanden Teammeetings statt, die von Justus geleitet wurden. Diese Treffen dienten der Koordination, der Besprechung des Fortschritts und der Zuteilung neuer Aufgaben. Es gab wenige Ausnahmen von diesem Rhythmus.
- **Aufgabenverteilung:** Justus und Jonas W. erstellten wöchentlich ein Excel-Sheet mit klar definierten Aufgaben, die so fair wie möglich verteilt wurden.
- **Pflichtenheft:** Am Pflichtenheft arbeiteten alle Teammitglieder gemeinsam, um sicherzustellen, dass alle Anforderungen umfassend berücksichtigt wurden.
- **Entwicklungsprozess:**
  - Die Backend-Fundamente wurden von Phillip und Jonas geschaffen.
  - Frontend-Verbesserungen wurden insbesondere von Daniel R. und Qipeng durchgeführt.
  - Fehlerbehebungen und technische Fragen wurden hauptsächlich von Phillip, Justus und Basti bearbeitet.
- **Testing:** Die Tests wurden von Basti und Daniel G. durchgeführt, um die Funktionalität und Qualität des Fremdsystems sicherzustellen.

2. Systemumgebung

2.1. Server Spezifikation

CPU	Dual-Core-Prozessor mit mindestens 2 GHz Taktfrequenz (basierend auf einer CPU-Auslastung von <1 % bei Idle-Betrieb und den Anforderungen einer Java-basierten Spring Boot-Anwendung).

Arbeitsspeicher (RAM)	Mindestens 2 GB, empfohlen 4 GB, da der Heap-Bereich während des Betriebs ca. 200 MB belegt, jedoch Spitzen bei komplexen Abfragen berücksichtigt werden müssen.
Festplattenspeicher	Mindestens 5 GB freier Speicherplatz, um die Anwendung, Logs und temporäre Dateien zu speichern.
Netzwerk	Stabile Verbindung mit mindestens 10 Mbps für interne Zugriffe und externe API-Aufrufe.

2.2. Software Spezifikation

Betriebssystem	Linux (Ubuntu 20.04 oder neuer empfohlen) oder ein vergleichbares, Java-kompatibles System.
Java	Java 21 oder höher, da Spring Boot auf den neuesten LTS-Versionen optimal funktioniert.
MySQL	MySQL 8.0 oder höher mit SSL-Unterstützung.

2.3. Client Anforderungen

Browser	Die aktuelle Version von den 0,2% meist genutzten Browser (Safari, Chrome, Edge, Firefox, Mozilla und Opera)
Gerät	Mobilgeräte oder Desktop-PCs mit mindestens 2 GB RAM und Dual-Core-CPU

3. Entwicklungsschritte

Die Entwicklung unseres Projekts wurde in klar definierten Phasen durchgeführt, um eine systematische und effiziente Bearbeitung sicherzustellen. Nachfolgend werden die wesentlichen Schritte und Meilensteine beschrieben, die während der Implementierungsphase erreicht wurden.

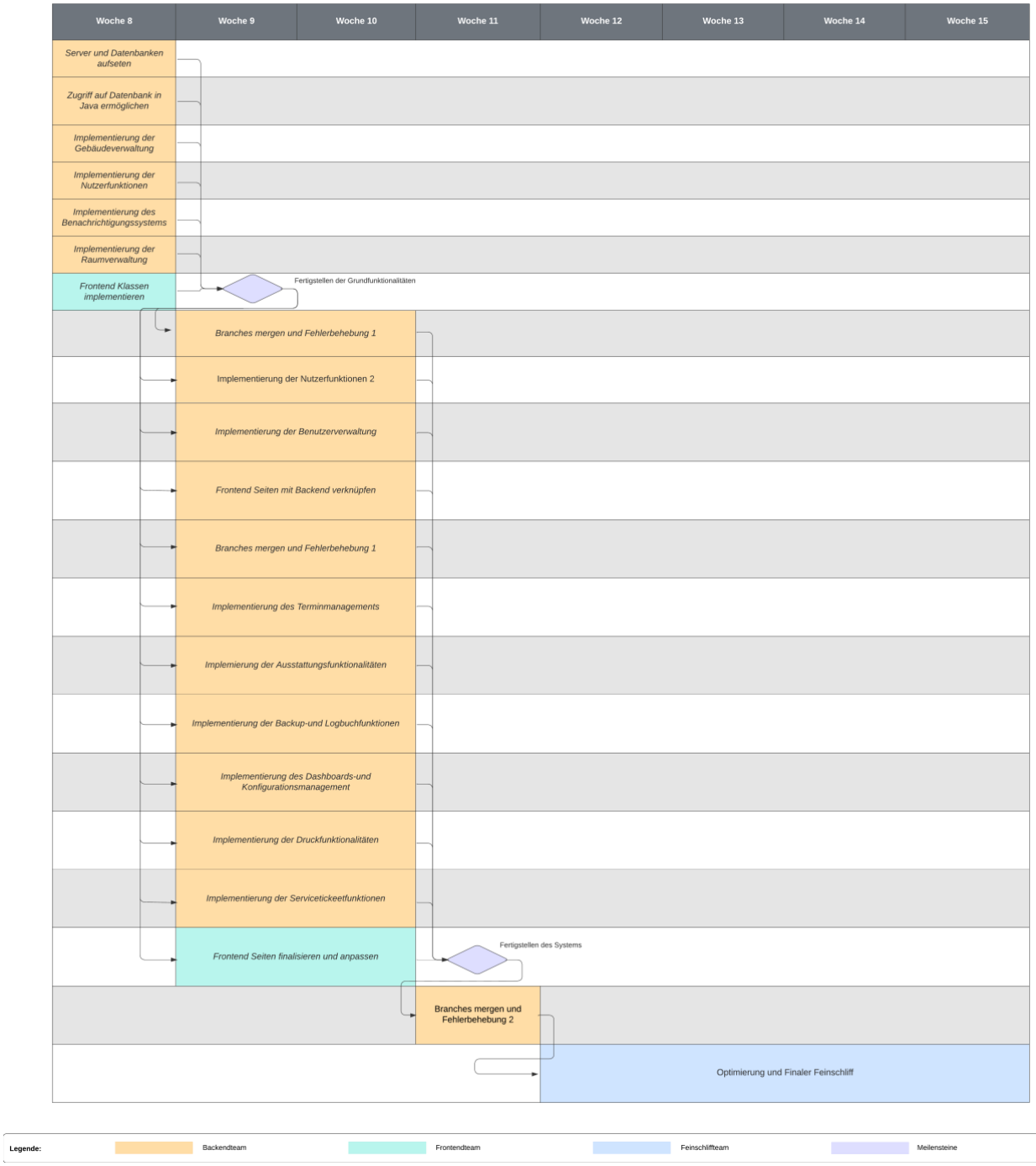

Abbildung 1: Gantt-Diagramm für Implementierungsphase

Die Projektplanung begann mit der Erstellung eines umfassenden Pflichtenhefts, in dem die Anforderungen und Ziele des Projekts detailliert beschrieben wurden. Nach Abschluss des Pflichtenhefts wurde ein Diagramm entwickelt, das die Abfolge der zu implementierenden Funktionen darstellte. Dieses Diagramm enthielt eine Priorisierung der Aufgaben sowie eine Schätzung des Zeitaufwands für jede Funktion.

Vor Beginn der Implementierungsphase hat sich jedes Teammitglied intensiv mit den Aspekten des zu entwickelnden Systems auseinanderzusetzen. Dies beinhaltete ein selbstständiges Studium der erforderlichen Technologien und Tools, wie z. B. Java Spring für das Backend, React für das Frontend sowie die Nutzung von MySQL für die Datenbankintegration. Ziel war es, ein gemeinsames Verständnis der Systemanforderungen und der zu verwendenden Technologien sicherzustellen.

Regelmäßige Teammeetings (donnerstags um 18:00 Uhr) wurden eingeführt, um den Fortschritt zu überwachen, offene Fragen zu klären und die Aufgabenverteilung auf Basis des erstellten Diagramms vorzunehmen. Ein wöchentlich aktualisiertes Aufgabenblatt unterstützte die Organisation und Transparenz innerhalb des Teams, dieses wurde in Miro angefertigt.

4. Versionsverwaltung

Die Versionsverwaltung unseres Projekts wird mithilfe von **Git** und **GitHub** organisiert. Die Branching-Strategie und Arbeitsweise sorgen für eine klare Trennung der Entwicklungsaufgaben und gewährleisten Stabilität im Hauptprojekt.

4.1. Branch-Organisation

Die Branches im Projekt sind in verschiedene Kategorien unterteilt, um die Entwicklung zu strukturieren und parallele Arbeiten zu erleichtern:

- **main-Branch**: Enthält stets die stabilste und produktionsbereite Version des Projekts.
- **develop-Branch**: Dient als Hauptentwicklungszweig. Hier werden abgeschlossene Features integriert, bevor sie in den main-Branch gemerged werden.
- **Feature-Branches**: Für jede neue Funktion oder Aufgabe wird ein separater Feature-Branch erstellt, z. B.:
    - feature/Systemwartung
    - feature/Nutzerinteraktionen
    - feature/Gebäudeverwaltung-Raumverwaltung

Die Benennung der Feature-Branches folgt einem klaren Muster: **feature/<Beschreibung>**, das die Aufgabe des Branches beschreibt. Diese Struktur erleichtert die Nachverfolgbarkeit und Zusammenarbeit.

4.2. Arbeitsweise

1. **Erstellen eines Feature-Branches**: Entwickler erstellen einen neuen Branch aus develop, um Änderungen für ein spezifisches Feature vorzunehmen.
2. **Entwicklung im Feature-Branch**: Die Arbeiten werden lokal durchgeführt, regelmäßig committet und gepusht.
3. **Pull Requests (PR)**: Sobald das Feature abgeschlossen ist, wird ein Pull Request erstellt, um die Änderungen in den develop-Branch zu integrieren.
4. **Code Reviews**: Andere Teammitglieder prüfen den Code und geben Feedback, bevor der Merge in develop erfolgt.
5. **Release**: Nach der Integration und erfolgreichen Tests im develop-Branch wird der Code in den main-Branch gemerged, um ein neues Release zu erstellen.

4.3. Branch-Übersicht

Hier ist eine aktuelle Übersicht der Branches in unserem Repository:

- Hauptbranches:
    - main
    - develop
- Feature-Branches:
    - feature/Systemwartung
    - feature/Kalender-Benachrichtigung-Terminmanagement

    - feature/Nutzerinteraktionen
    - feature/Gebäudeverwaltung-Raumverwaltung
- Spezielle Branches:
    - frontend-verbinden: Enthält Arbeiten zur Integration des Frontends.
    - Optimierung: Dient zur Verbesserung von Performance und Codequalität.

4.4. Zusammenarbeit und Verantwortlichkeiten

- **Feature-Entwicklung**: Entwickler arbeiten in ihren jeweiligen Feature-Branches.
- **Code-Qualität**: Pull Requests und Code Reviews gewährleisten eine hohe Codequalität.
- **Branch-Management**: Nur ausgewählte Teammitglieder sind berechtigt, Änderungen in den main-Branch zu übernehmen, um Stabilität zu garantieren.

4.5. Fazit

Durch die strukturierte Organisation der Branches und die konsequente Nutzung von GitHub als Versionskontrollsystem konnten wir die Zusammenarbeit im Team erheblich verbessern und sicherstellen, dass das Projekt stabil und gut dokumentiert bleibt.

5. Projektstruktur

Dieses Kapitel beschreibt die technische Struktur des Projekts, inklusive der verwendeten Frameworks, Design-Patterns und Best-Practices, die zur Entwicklung der Anwendung eingesetzt wurden.

Die Projektstruktur ist nach den Prinzipien einer **mehrschichtigen Architektur** aufgebaut, um eine klare Trennung der Verantwortlichkeiten zu gewährleisten. Die wichtigsten Schichten sind:

- **Controller-Schicht:** Beinhaltet die REST-Controller, die Anfragen vom Frontend entgegennehmen und passende Services aufrufen.

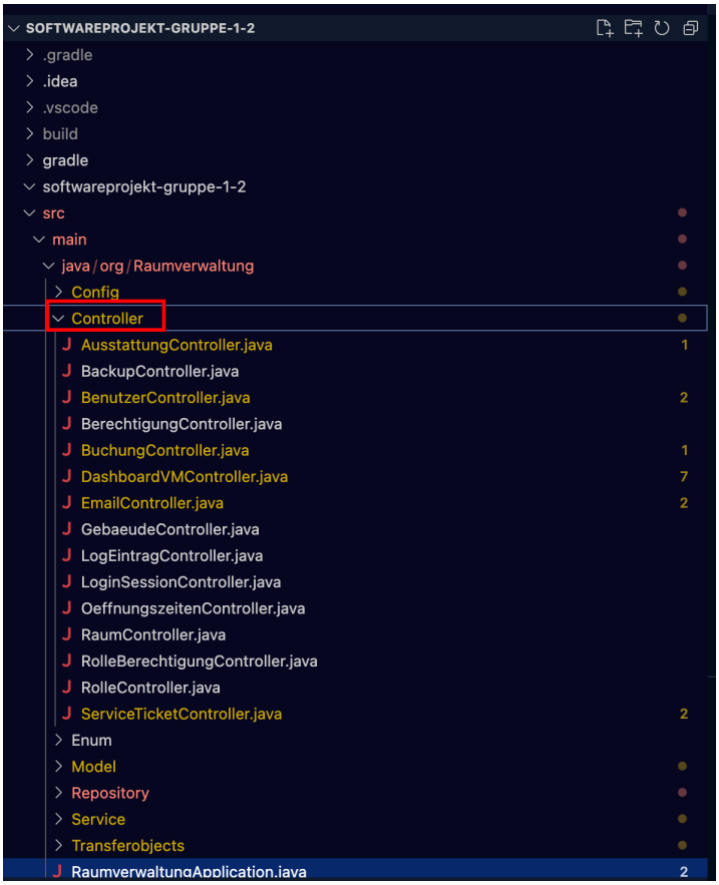

Abbildung 2: Controller-Ordner

- **Service-Schicht:** Implementiert die Geschäftslogik und delegiert Datenbankoperationen an die Repository-Schicht.

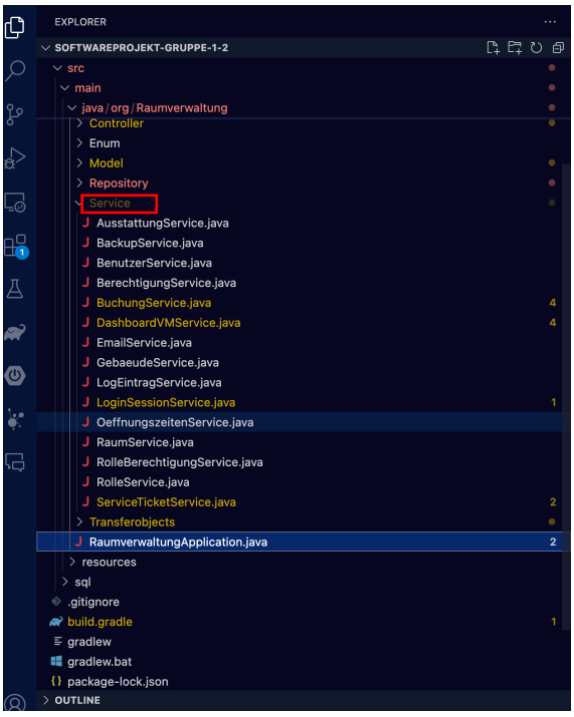

Abbildung 3: Service-Ordner

- **Repository-Schicht:** Verantwortlich für den Datenzugriff. Nutzt JPA/Hibernate zur Kommunikation mit der Datenbank.

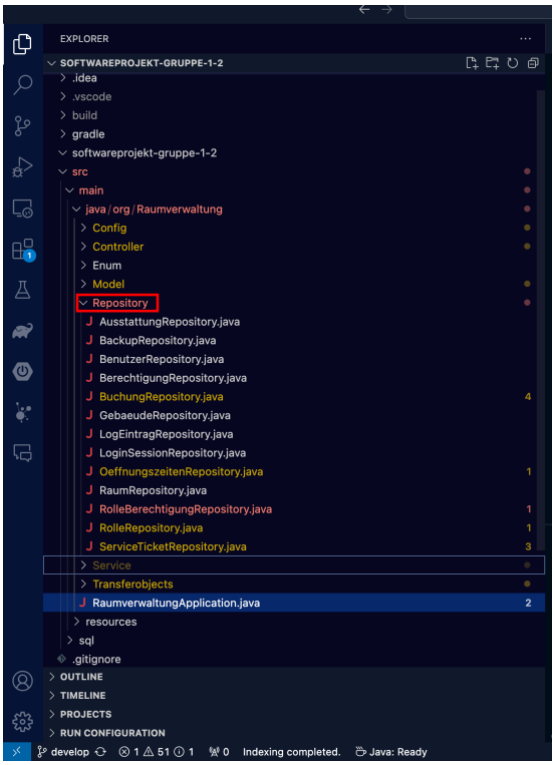

Abbildung 4: Repository Ordner

- **Model-Schicht:** Repräsentiert die Domänenobjekte (JPA-Entitäten), die mit der Datenbank verknüpft sind.

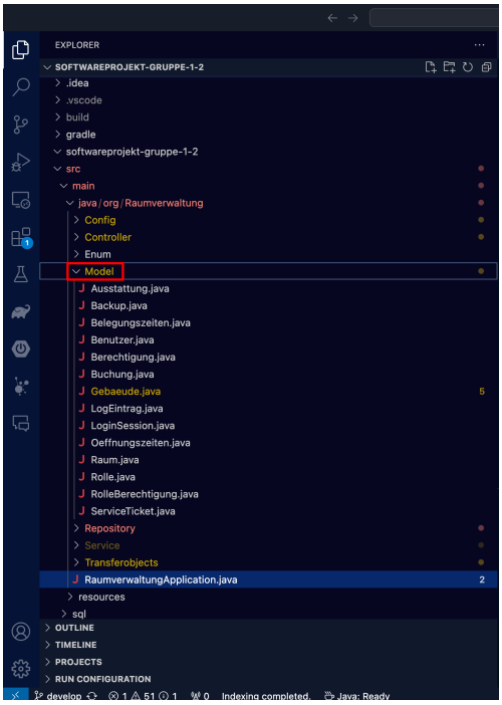

Abbildung 5: Model Ordner

- **DTO-Schicht (Transfer Objects):** Definiert Objekte, die zwischen Frontend und Backend ausgetauscht werden.

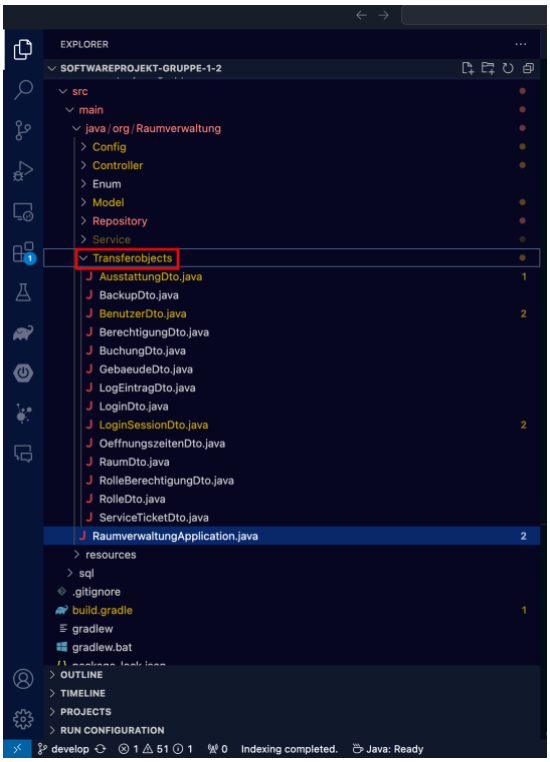

Abbildung 6: DTO Ordner

- Enum: Repräsentiert vorangefertigte Enum-Klassen

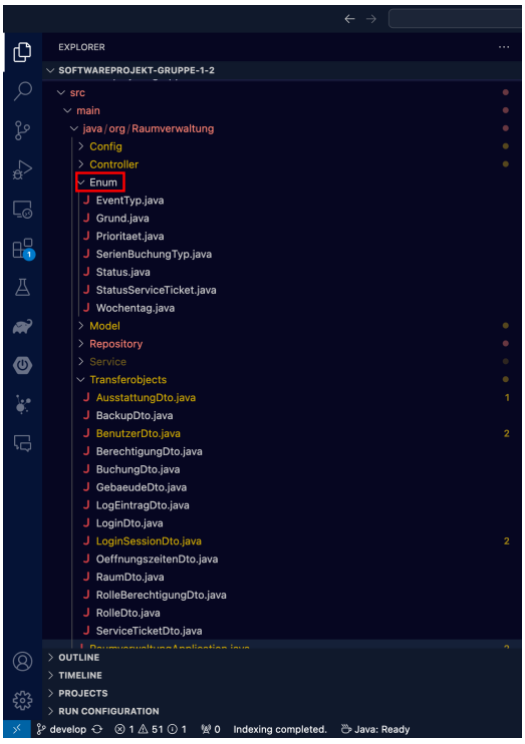

Abbildung 7: Enum Ordner

- **Frontend-Schicht:** Implementiert mithilfe von React die Benutzeroberfläche, die mit den REST-APIs des Backends kommuniziert.
    - Node_modules: enthält installiert node.js Module
    - Public: enthält Bilder die in unsere Applikation verwendet worden sind
    - Src: Stellt die Hauptkomponente da in ihr ist noch ein Ordner **assets** enthalten der die Button Designs enthält ,**components** enthält die Header Komponente für unser Applikation, und die Buttons (zweit genanntes wird noch in einigen CSS-Klassen modifiziert). Der Ordner pages enthält für alle Seiten jeweils eine js-Klasse und eine CSS Klasse. Der Ordner services enthält jeweils zwei Klassen um die Gebäudedaten und Benutzerdaten zu fetchen

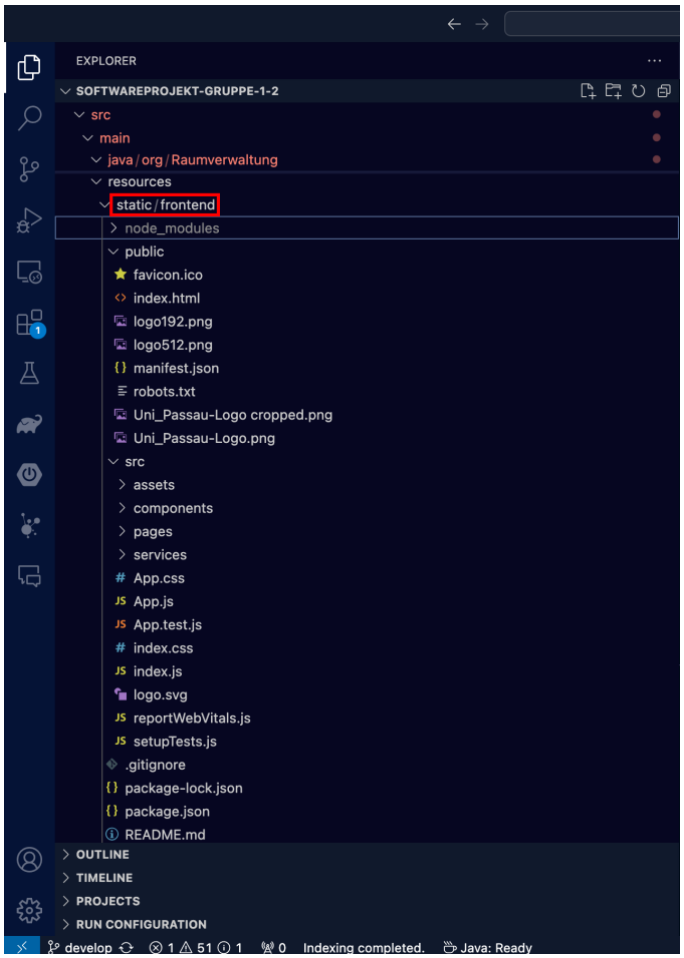

Abbildung 8: Frontend Ordner

5.1. Framework

Das Projekt basiert auf mehreren Frameworks und Technologien, die die Entwicklung und Wartung der Anwendung erleichtern.

Spring Boot

- **Beschreibung:** Spring Boot ist das zentrale Framework des Projekts. Es bietet eine einfache und schnelle Möglichkeit, Spring-Anwendungen zu entwickeln, indem es die Konfiguration minimiert.

- **Funktionen:**
  - REST-Schnittstellen (Spring Web).
  - Datenbankzugriff (Spring Data JPA).
  - Validierung (Spring Validation, Jakarta Validation).

Hibernate (JPA)

- **Beschreibung:** Hibernate wird zur Datenbankkommunikation genutzt. Es ermöglicht Object-Relational Mapping (ORM), sodass Java-Objekte direkt in Datenbanktabellen gespeichert und abgerufen werden können.
- **Funktionen:**
  - Automatische Generierung von SQL-Abfragen.
  - Verwaltung von Entitäten und deren Beziehungen (z. B. `@OneToMany`, `@ManyToOne`).

Modelmapper

- **Beschreibung:** Ein Framework, das für die Konvertierung von Entitäten in DTOs und umgekehrt verwendet wird.
- **Funktion:** Reduziert Boilerplate-Code für Mapping-Operationen.

React

- **Beschreibung:** React ist ein JavaScript-Framework zur Erstellung von Benutzeroberflächen. Es wird für die Entwicklung der Frontend-Schicht verwendet.
- **Funktion:**
  - Erstellung von wiederverwendbaren UI-Komponenten.
  - Zustandshandhabung mit Hooks (z. B. `useState`, `useEffect`).
  - Kommunikation mit dem Backend über Fetch- oder Axios-APIs.

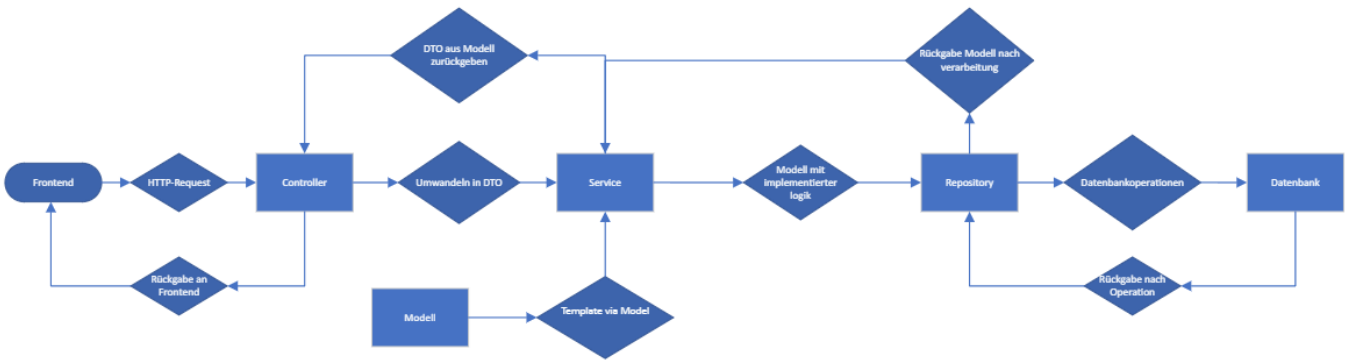

Abbildung 9: Flowchart Schichtenarchitektur

5.2. Pattern

Im Projekt werden mehrere bewährte Design-Patterns angewandt, um die Wartbarkeit und Erweiterbarkeit der Anwendung zu gewährleisten.

Model-View-Controller

- **Beschreibung:** Das MVC-Pattern wird verwendet, um die Anwendung in drei Schichten zu unterteilen:
- **Model:** Enthält die JPA-Entitäten.
- **Controller:** Bereitstellung der REST-Endpunkte.
- **Service:** Implementiert die Geschäftslogik.

Repositories

- **Beschreibung:** Das Repository-Pattern abstrahiert den Zugriff auf die Datenbank.
- **Vorteile:**
    - Wiederverwendbare und testbare Datenzugriffsmethoden.
    - Einfaches Erstellen von benutzerdefinierten Abfragen (z. B. `@Query`).

Data Transfer Objects (Kurz DTO)

- **Beschreibung:** DTOs werden verwendet, um Daten zwischen dem Frontend und Backend auszutauschen.
- **Vorteile:**
    - Verhinderung der direkten Exposition von Entitäten.
    - Sicherheit und Flexibilität bei der Datenübertragung.

Component Based Architecture (Frontend)

- **Beschreibung:** Im Frontend werden UI-Komponenten als wiederverwendbare und eigenständige Bausteine entwickelt.
- **Vorteile:**
    - Klare Trennung von Verantwortlichkeiten.
    - Bessere Wiederverwendbarkeit und Lesbarkeit des Codes.

5.3. Best Practices

Im Projekt wurden verschiedene Best Practices angewendet, um die Codequalität und Wartbarkeit zu optimieren.

Schichtenarchtitektur

- **Beschreibung:** Die Trennung der Verantwortlichkeiten in Controller-, Service- und Repository-Schichten erleichtert die Wartung und Testbarkeit des Codes.
- **Vorteile:**
    - Klare Zuständigkeiten.
    - Leichtere Fehlersuche und Erweiterung.

Input Validation

- **Beschreibung:** Eingaben werden mithilfe von Jakarta Validation (`@Valid`) validiert, z. B. in DTO-Klassen
- **Vorteil:** Verhindert ungültige Daten bereits im Controller.

Ausnahmebehandlung

- **Beschreibung:** Exceptions werden durch zentrale Fehlerbehandlung abgefangen (z. B. `@ExceptionHandler` in globalen Exception-Handler-Klassen).
- **Vorteil:** Einheitliche Fehlerbehandlung und bessere Benutzererfahrung.

Frontend Komponenten

- **Beschreibung:** Exceptions werden durch zentrale Fehlerbehandlung abgefangen (z. B. `@ExceptionHandler` in globalen Exception-Handler-Klassen).
- **Vorteil:** Einheitliche Fehlerbehandlung und bessere Benutzererfahrung.

6. Softwarekomponenten

Das Projekt verwendet das **Schichtenarchitektur-Pattern** (Layered Architecture Pattern), das in vielen Unternehmensanwendungen weit verbreitet ist. Dieses Pattern teilt die Anwendung in verschiedene Schichten, die jeweils eine spezifische Verantwortung haben. Die Hauptschichten in diesem Projekt sind:

Abbildung 10: Ordnerstruktur

Die von uns entwickelte Web-Applikation besteht im Wesentlichen aus drei Komponenten:

- Dem Backend (/project-root/src/main/java/org/Raumverwaltung) entwickelt mit Java Spring
- Dem Frontend ((/project-root/src/main/resources/static/frontend/src) entwickelt mit React.js
- Der MySQL Datenbank „Raumverwaltung“ (SQL)

Die folgende Dokumentation gibt einen kurzen Überblick auf den Aufbau der 3 Komponenten, um die Navigation durch die Ordnerstruktur Außenstehenden zu erleichtern. Außerdem soll die Funktionsweise der Applikation aufgezeigt werden, um leichteres Verständnis zu gewährleisten.

6.1. Das Java Spring Backend

Das Backend ist eine Spring Boot-Anwendung, die das Schichtenarchitektur-Pattern verwendet. Die Schichtenarchitektur ist ein Designprinzip, das die Anwendung in logisch getrennte Schichten unterteilt. Jede Schicht übernimmt dabei eine spezifische Rolle:

1. **Controller-Schicht**: Diese Schicht ist für die Verarbeitung von HTTP-Anfragen und das Zurückgeben von HTTP-Antworten verantwortlich. Sie enthält die Controller-Klassen, die die Endpunkte definieren.

2. **Service-Schicht**: Diese Schicht enthält die Geschäftslogik der Anwendung. Sie verarbeitet die Daten, die von der Controller-Schicht empfangen werden, und führt die notwendigen Operationen aus.

3. **Repository-Schicht**: Diese Schicht ist für den Datenzugriff verantwortlich. Sie enthält die Repository-Klassen, die die Datenbankoperationen durchführen.

4. **Model-Schicht**: Diese Schicht enthält die Entitätsklassen, die die Datenbanktabellen repräsentieren.

5. **DTO-Schicht (Data Transfer Objects)**: Diese Schicht enthält die DTO-Klassen, die verwendet werden, um Daten zwischen den Schichten zu übertragen.

6. **Konfigurations-Schicht**: Diese Schicht enthält die Konfigurationsklassen, die die Anwendung konfigurieren, z.B. Sicherheitskonfigurationen.

Beispiel:

Ein Benutzer möchte eine Raumreservierung vornehmen. Die Anfrage wird an die Controller-Schicht gesendet, die die Service-Schicht aufruft. Dort wird geprüft, ob der Raum verfügbar ist. Die Repository-Schicht speichert die Reservierung in der MySQL-Datenbank, und das Ergebnis wird über die DTO-Schicht an das Frontend übermittelt

6.2. Das React.js Frontend

Das Frontend ist eine React-Anwendung, die aus Seitenkomponenten und wiederverwendbaren Komponenten besteht. Hier wurden noch die Tools Material UI und Emotion verwendet. Material UI erleichtert die Erstellung eines konsistenten Designs und spart Entwicklungszeit, während Emotion eine nahtlose Integration von Stilen direkt in React-Komponenten ermöglicht. Die React App greift jeweils auf beide Tools zu.

6.3. Die MySQL Datenbank

Die MySQL-Datenbank dient als zentrale Komponente für die Speicherung und Verwaltung aller Anwendungsdaten in unserem System „Raumverwaltung“. Sie wurde aufgrund ihrer Zuverlässigkeit, hohen Performance und einfachen Integration in Java-basierte Backends wie das von uns verwendete Spring Framework ausgewählt.

Um die Sicherheit der Datenbank zu gewährleisten, wurden folgende Maßnahmen implementiert:

- **Benutzer-Authentifizierung**: Nur autorisierte Anwendungen können auf die Datenbank zugreifen.
- **Verschlüsselte Verbindungen**: Kommunikation zwischen Backend und Datenbank erfolgt über SSL.

Vorteile der MySQL-Datenbank

- **Hohe Skalierbarkeit**: Geeignet für wachsende Datenmengen und Nutzeranforderungen.
- **Einfacher Zugriff**: MySQL bietet eine intuitive Schnittstelle für Datenbankadministration und Abfragen.
- **Open-Source-Lizenz**: Minimiert die Kosten bei gleichzeitigem Zugang zu einer breiten Entwickler-Community.

Datenbankschema

Die Struktur unserer Datenbank ist in verschiedene Tabellen unterteilt, die die unterschiedlichen Komponenten des Raumverwaltungssystems repräsentieren. Die folgende Abbildung zeigt das Datenmodell:

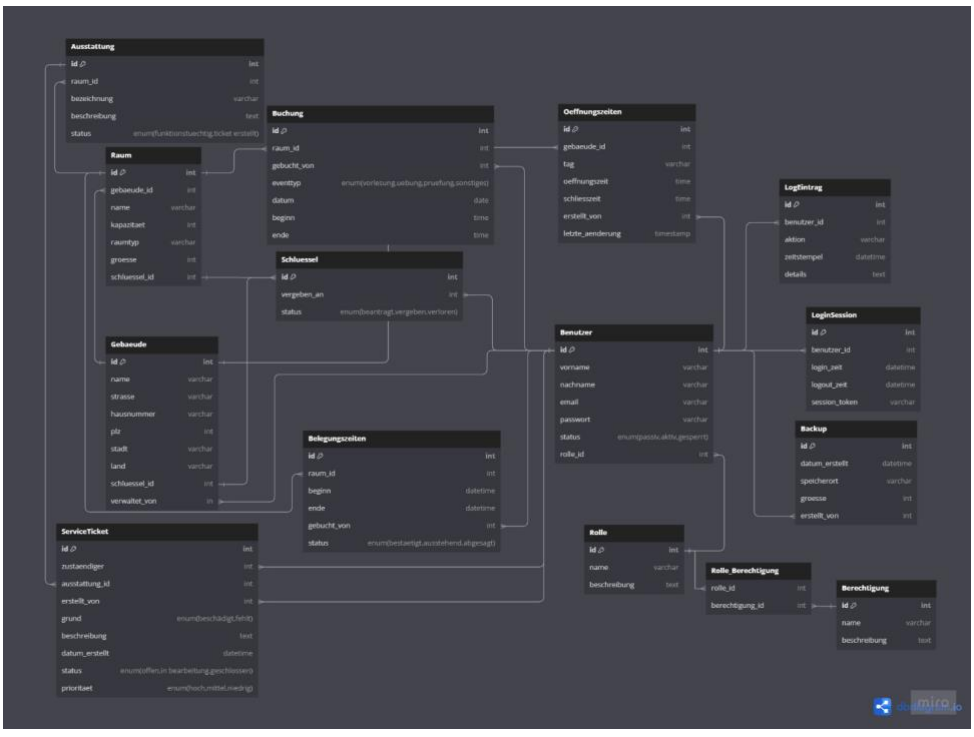

Abbildung 11: Datenbankschema

Die wichtigsten Tabellen und deren Funktionen sind im Folgenden beschrieben:

- **Benutzer**: Speichert die registrierten Nutzer mit Attributen wie Name, E-Mail, Passwort und Status. Nutzer können verschiedene Rollen zugewiesen bekommen.
- **Rolle**: Definiert verschiedene Nutzerrollen innerhalb des Systems (z. B. Administrator, Standardbenutzer).
- **Rolle_Berechtigung / Berechtigung**: Verknüpft Rollen mit spezifischen Zugriffsrechten.
- **Raum**: Enthält Informationen zu allen verwalteten Räumen, einschließlich Kapazität und zugehörigem Gebäude.
- **Gebäude**: Speichert Informationen über die verschiedenen Gebäude, in denen Räume verwaltet werden.
- **Buchung**: Dokumentiert Raumreservierungen, einschließlich Buchungszeit, Nutzer und Eventtyp.
- **Belegungszeiten**: Speichert spezifische Zeiträume, in denen ein Raum belegt ist.
- **Öffnungszeiten**: Definiert die Betriebszeiten der Gebäude, um Zugangsberechtigungen und Buchungen zu regeln.
- **Ausstattung**: Speichert Informationen zu technischen Geräten und Möbeln, die in den Räumen verfügbar sind.
- **ServiceTicket**: Wird genutzt, um Defekte oder fehlende Ausstattung zu melden und deren Bearbeitungsstatus zu verfolgen.
- **LogEintrag**: Zeichnet alle sicherheitsrelevanten Aktionen auf, z. B. Login-Versuche oder Datenänderungen.
- **LoginSession**: Speichert aktive und vergangene Sitzungen von Nutzern zur Nachverfolgung von Anmeldevorgängen.
- **Backup**: Protokolliert erstellte Sicherungen der Datenbank mit Speicherort und Zeitstempel.

Diese relationale Struktur ermöglicht eine effiziente Speicherung und Abfrage von Daten, während sie gleichzeitig die Datenintegrität und Sicherheit gewährleistet. Sie stellt sicher, dass alle Benutzer nur auf die für sie relevanten Daten zugreifen können und erleichtert gleichzeitig die Verwaltung von Räumen, Buchungen und Berechtigungen innerhalb der Anwendung. Im Verzeichnis Archiv/Dokumentation/Tools und verwendete Technologien/ befindet sich noch ein UML-Klassendiagramm um das Zusammenwirken der Funktionen zu veranschaulichen.

6.4. Allgemeine Funktionen

Die **Controller-Klassen** und **Konfigurationsklassen** bilden das zentrale Steuerungselement der Geschäftslogik des Systems. Während die Controller die Schnittstelle zwischen Frontend und Backend darstellen und die Kernprozesse wie Authentifizierung, Buchungsmanagement und Berechtigungsverwaltung orchestrieren, sorgen die Konfigurationsklassen für eine sichere und effiziente Umgebung, indem sie zentrale Funktionen wie Fehlerbehandlung, Sicherheitsrichtlinien und systemweite Einstellungen bereitstellen. Durch diese Struktur wird die grundlegende Funktionsweise des Systems ersichtlich, ohne jede einzelne Service-, Model- oder Repository-Schicht im Detail erläutern zu müssen

Config

AppConfiguration.java	• @Configuration: Markiert die Klasse als Konfigurationsklasse. • @Bean: Markiert die Methode, die eine Bean definiert. • modelMapper(): Erstellt und konfiguriert eine ModelMapper Instanz.
GlobalExceptionHandler.java	• @ControllerAdvice: Markiert die Klasse als globalen Ausnahmebehandler, der für alle Controller in der Anwendung gilt. • @ExceptionHandler(ResponseStatusException.class): Markiert die Methode als Behandler für ResponseStatusException. • handleResponseStatusException(ResponseStatusException ex): Methode, die ResponseStatusException behandelt, indem sie eine strukturierte Antwort mit Statuscode, Fehlermeldung und Zeitstempel zurückgibt.
SecurityConfiguration.java	• @Configuration: Kennzeichnet die Klasse als Konfigurationsklasse, die Bean-Definitionen enthält. • @Bean: Kennzeichnet eine Methode, die eine Bean-Definition bereitstellt, die im Spring Application Context registriert wird. • securityFilterChain(HttpSecurity http): Konfiguriert die Sicherheitsfilterkette, einschließlich CORS-Einstellungen, um Anfragen von bestimmten Ursprüngen und mit bestimmten Methoden und Headern zu erlauben. • passwordEncoder(): Definiert einen PasswordEncoder Bean, der für die sichere Speicherung von Passwörtern verwendet wird.

Controller

AusstattungsController.java	• create(AusstattungDto ausstattungDto): Erstellt eine neue Ausstattung und ruft ausstattungService.create(ausstattungDto) auf.

	• update(Long id, AusstattungDto ausstattungDto): Aktualisiert eine bestehende Ausstattung und ruft ausstattungService.update(id, ausstattungDto) auf. • delete(Long id): Löscht eine Ausstattung anhand der ID und ruft ausstattungService.delete(id) auf.
BackupController.java	• createBackup(BackupDto backupDto): Erstellt ein neues Backup und ruft backupService.scheduleBackup(backupDto) auf. Behandelt mögliche Fehler und gibt entsprechende HTTP-Statuscodes zurück. • getAllBackups(): Holt alle Backups und ruft backupService.getAllBackups() auf. • deleteBackup(Long id): Löscht ein Backup anhand der ID und ruft backupService.deleteBackup(id) auf. • restoreBackup(String backupFilePath): Stellt ein Backup wieder her und ruft backupService.restoreBackup(backupFilePath) auf. Behandelt mögliche IOException.
BenutzerController.java	• create: Erstellt einen neuen Benutzer. • createMitRolle: Erstellt einen neuen Benutzer mit einer bestimmten Rolle. • deleteById: Löscht einen Benutzer anhand der ID. • update: Aktualisiert die Daten eines bestehenden Benutzers. • getAllBenutzer: Ruft alle Benutzer ab. • getBySessionToken: Ruft einen Benutzer anhand des Session-Tokens ab. • getById: Ruft einen Benutzer anhand der ID ab. • getFacilityManagers: Ruft alle Facility Manager ab. • changePassword: Ändert das Passwort eines Benutzers. • getByEmail: Ruft einen Benutzer anhand der E-Mail-Adresse ab.
BerechtigungController.java	• getAll: Ruft alle Rollen-Berechtigungs-Zuweisungen ab. • getById: Ruft eine spezifische Rollen-Berechtigungs-Zuweisung anhand der ID ab. • create: Erstellt eine neue Rollen-Berechtigungs-Zuweisung. • update: Aktualisiert eine bestehende Rollen-Berechtigungs-Zuweisung. • delete: Löscht eine Rollen-Berechtigungs-Zuweisung. • getByRolle: Ruft Rollen-Berechtigungs-Zuweisungen basierend auf der Rolle ab.
BuchungController.java	• getBuchungenByGebuchtVon: Ruft alle Buchungen eines bestimmten Benutzers ab. • getAllBenutzer: Ruft alle Buchungen ab. • getBuchungen: Ruft Buchungen eines Raums zwischen zwei Daten ab. • getKalenderwoche: Ruft Buchungen einer Kalenderwoche ab. • getBuchungenByRaum: Ruft alle Buchungen eines bestimmten Raums ab.

	• createBuchung: Erstellt eine neue Buchung. • updateBuchung: Aktualisiert eine bestehende Buchung. •
DashboardVMController.java	• getRaumauslastung(String startDate, String endDate): Berechnet die Raumauslastung für einen bestimmten Zeitraum. • durchschnittlicheRaumauslastungHeute(): Berechnet die durchschnittliche Raumauslastung für heute. • getOffeneServiceTickets(): Zählt die offenen Service-Tickets. • getRaumbuchungen(String startDate, String endDate): Zählt die Raumbuchungen für einen bestimmten Zeitraum. • getRaumbuchungenWoche(): Zählt die Raumbuchungen für die aktuelle Woche. • getRaumbuchungenMonat(): Zählt die Raumbuchungen für den aktuellen Monat.

EmailController.java	• sendEmail(String email): Sendet eine E-Mail zum Zurücksetzen des Passworts an die angegebene E-Mail-Adresse.
	  ○ Sucht den Benutzer anhand der E-Mail-Adresse.
	  ○ Generiert einen Link zum Zurücksetzen des Passworts.
	  ○ Sendet eine E-Mail mit dem Link an den Benutzer.

GebaeudeController.java	• create(GebaeudeDto gebaeudeDTO): Erstellt ein neues Gebäude. • deleteById(Long id): Löscht ein Gebäude anhand der ID. • update(Long id, GebaeudeDto gebaeudeDto): Aktualisiert ein bestehendes Gebäude. • getAllGebaeude(): Ruft alle Gebäude ab. • getById(Long id): Ruft ein Gebäude anhand der ID ab. • getFacilityManagerByBuildingId(Long id): Ruft den Facility Manager eines Gebäudes ab.
LogEintragController.java	• getLogEintraegeForBenutzer(Long benutzerId): Holt alle Log-Einträge für einen bestimmten Benutzer. • getLogEintraegeForAktion(String aktion): Holt alle Log-Einträge für eine bestimmte Aktion. • getLogEintraegeImZeitraum(String start, String end): Holt alle Log-Einträge innerhalb eines bestimmten Zeitraums. • getLogEintraegeLetzte30Tage(): Holt die Log-Einträge der letzten 30 Tage. • exportiereLogEintraegeAlsCsv(String start, String end): Exportiert die gefilterten Logeinträge als CSV-Datei. • protokolliereAktion(Long benutzerId, String aktion, String details): Protokolliert eine Aktion, die von einem Benutzer ausgeführt wurde. • getAlleLogEintraege(): Holt alle Log-Einträge.
LoginSessionController.java	• login(LoginDto loginDto): Loggt einen Benutzer ein und gibt die Sitzung zurück. • logout(String token): Loggt einen Benutzer aus und beendet die Sitzung.

OeffnungszeitenController.java	• getOeffnungszeitenByGebaeudeId(Long gebaeudeId): Ruft die Öffnungszeiten eines Gebäudes ab. • create(List<OeffnungszeitenDto> dto): Erstellt neue Öffnungszeiten. • delete(Long id): Löscht Öffnungszeiten anhand der ID. • updateOpeningHours(List<OeffnungszeitenDto> openingHours): Aktualisiert Öffnungszeiten.
RaumController.java	• create(RaumDto dto): Erstellt einen neuen Raum. • update(Long id, RaumDto dto): Aktualisiert einen bestehenden Raum. • delete(Long id): Löscht einen Raum anhand der ID. • getByGebaeude(Long gebaeudeId): Ruft alle Räume eines Gebäudes ab. • getRaumById(Long id): Ruft einen Raum anhand der ID ab. • searchRaeume(String name, Integer minKapazitaet, String ausstattung): Sucht nach Räumen basierend auf den angegebenen Kriterien.
RolleBerechtigungController.java	• getAll: Ruft alle Rollen-Berechtigungs-Zuweisungen ab. • getById: Ruft eine spezifische Rollen-Berechtigungs-Zuweisung anhand der ID ab. • create: Erstellt eine neue Rollen-Berechtigungs-Zuweisung. • update: Aktualisiert eine bestehende Rollen-Berechtigungs-Zuweisung. • delete: Löscht eine Rollen-Berechtigungs-Zuweisung. • getByRolle: Ruft Rollen-Berechtigungs-Zuweisungen basierend auf der Rolle ab.
RolleController.java	• getOeffnungszeitenByGebaeudeId: Ruft Öffnungszeiten eines Gebäudes ab. • create: Erstellt neue Öffnungszeiten. • delete: Löscht Öffnungszeiten anhand der ID. • updateOpeningHours: Aktualisiert Öffnungszeiten.
ServeTicketController.java	• create: Erstellt ein neues Service-Ticket. • update: Aktualisiert ein bestehendes Service-Ticket. • delete: Löscht ein Service-Ticket anhand der ID. • getAllServiceTickets: Ruft alle Service-Tickets ab. • getById: Ruft ein Service-Ticket anhand der ID ab. • deleteByRaumId: Löscht alle Service-Tickets eines Raums. • searchServiceTickets: Sucht nach Service-Tickets basierend auf Suchkriterien. • getByRoomAndEquipment: Ruft Service-Tickets basierend auf Raum und Ausstattung ab. • closeOpenTickets: Schließt offene Tickets mit einer bestimmten Ausstattung.

7. Eingesetzte Technologien und Strukturen der Software

Das Gebäudeverwaltungs- und Raumbuchungssystem basiert auf einer modernen Webtechnologie-Architektur, die eine benutzerfreundliche und plattformunabhängige Nutzung ermöglicht. Es wurden

verschiedene Tools und Frameworks eingesetzt, um die spezifischen Anforderungen des Projekts zu erfüllen und eine hohe Effizienz und Wartbarkeit sicherzustellen.

7.1. Frontend-Technologien

- React: Als Hauptframework für das Frontend wurde React verwendet, um eine dynamische und interaktive Benutzeroberfläche zu erstellen. React ermöglicht eine modulare Struktur der Komponenten und sorgt für eine optimale Performance.
- Material UI: Zur Gestaltung des Frontends wurde Material UI als Komponentenbibliothek verwendet. Diese bietet vorgefertigte und anpassbare UI-Komponenten, die für ein konsistentes und ästhetisches Design sorgen.
  Wir haben dieses Header-Design mithilfe von Material UI erstellt:

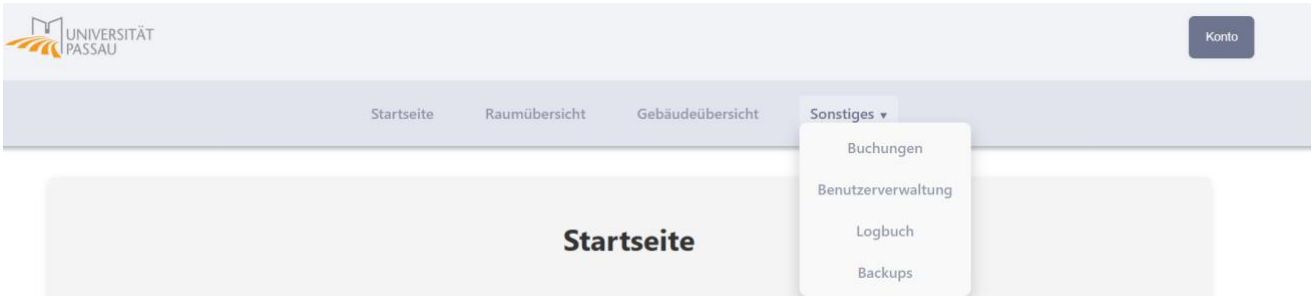

Abbildung 12: Header Componente

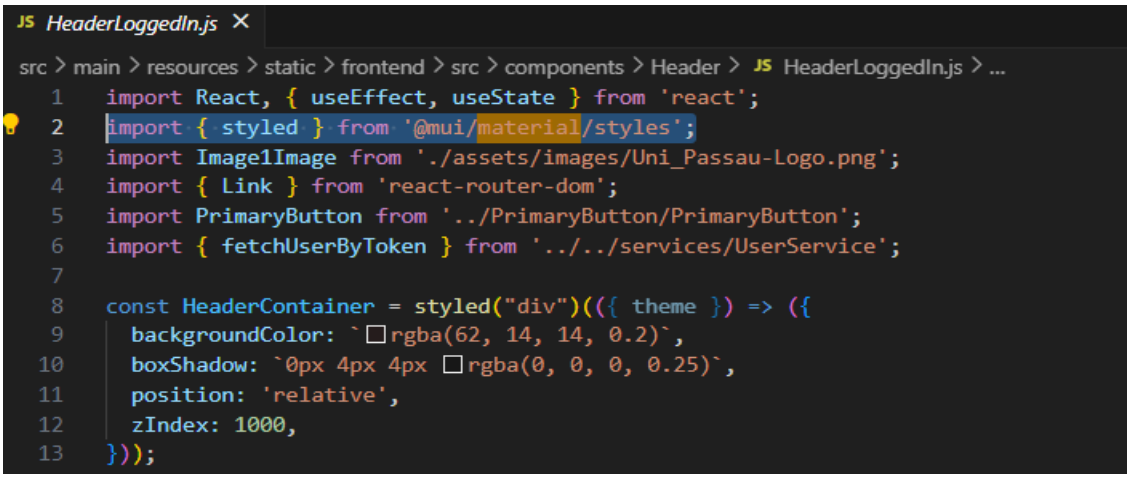

Abbildung 13: Code mit Material ui package

- Figma: Für die Erstellung von Mockups und Prototypen des Frontends wurde Figma verwendet. Das Tool erlaubte uns eine präzise Visualisierung des Designs und eine klare Kommunikation zwischen den Teammitgliedern bezüglich der Benutzeroberfläche.

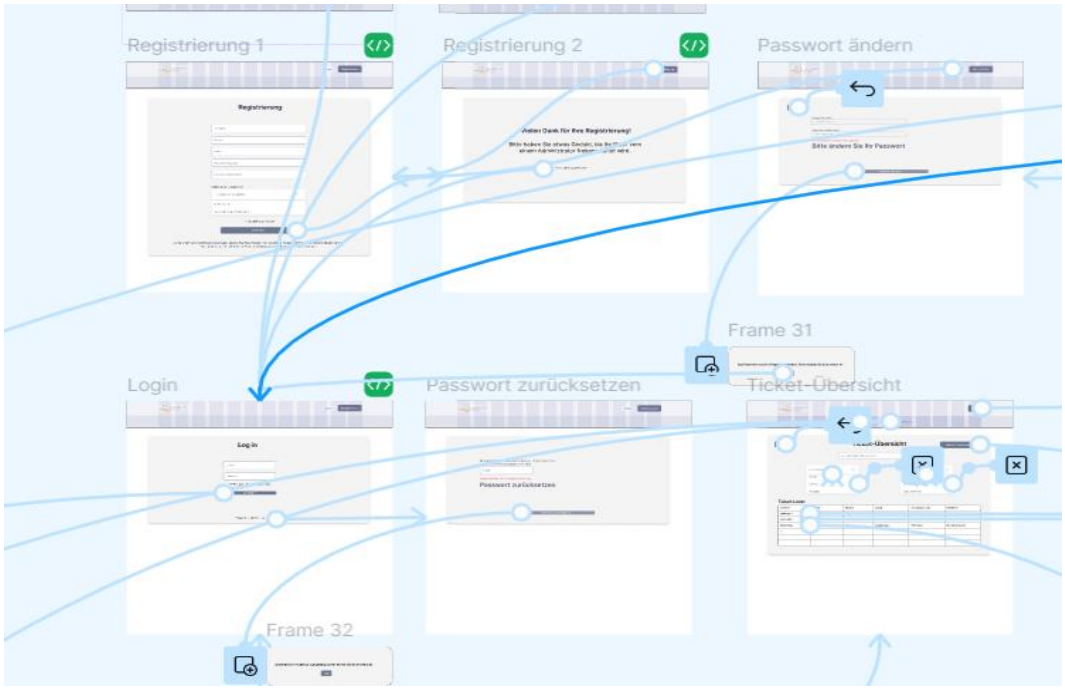

Abbildung 14: Figma Design

7.2. Backend-Technologien

- Java & Spring Framework: Das Backend wurde in Java unter Verwendung des Spring Frameworks implementiert. Spring bietet eine robuste Plattform für die Entwicklung von RESTful APIs, die den Datenaustausch zwischen Frontend und Backend ermöglichen.
- MySQL: Als Datenbankmanagementsystem wurde MySQL gewählt, um die Anwendungsdaten sicher und strukturiert zu speichern. MySQL zeichnet sich durch seine hohe Zuverlässigkeit und gute Integration mit Java-basierten Backends aus.

7.3. Systemumgebung und Infrastruktur

- Serverbetriebssystem: Das System läuft auf einem Linux-Ubuntu-Server der Universität, der eine sichere und stabile Umgebung für die Webanwendung bietet.
- Netzwerkzugriff: Die Anwendung ist über das Universitätsnetzwerk zugänglich, wodurch Sicherheitsstandards eingehalten und ein geschützter Zugriff gewährleistet werden.
- Kompatibilität: Die Webanwendung wurde für die Nutzung auf allen gängigen Browsern (Chrome, Edge, Firefox, Safari) optimiert. Zusätzlich wurde sichergestellt, dass sie auf mobilen Geräten genauso flüssig funktioniert wie auf Desktops.

7.4. Architektur der Software

- Die Software ist als Client-Server-Modell aufgebaut:
- Frontend: Kommuniziert über REST-APIs mit dem Backend und bietet eine intuitive Benutzeroberfläche.
- Backend: Behandelt die Geschäftslogik, verarbeitet Datenanfragen und verwaltet die Verbindung zur MySQL-Datenbank.
- Datenbank: Speichert und organisiert alle relevanten Daten, wie Benutzerinformationen, Raumbuchungen und Gebäudeinformationen.

7.5. Sicherheitsmaßnahmen

- Zugriffsschutz: Das System implementiert ein mehrstufiges Rollenkonzept (Admin, Verwaltungsmitarbeitende, Lehrende, Facility Manager), das den Zugriff auf Funktionen entsprechend der Nutzerrolle beschränkt.
- Datenschutz: Sensible Daten werden verschlüsselt übertragen und auf dem Server sicher gespeichert.
- Backups: Regelmäßige Datensicherungen werden durchgeführt, um Datenverlust vorzubeugen.

7.6. Zusammenfassung der eingesetzten Tools

Eingesetzte Tools und Technologien	**Verwendungszweck**
React	Frontend-Framework
Material UI	Komponentenbibliothek für UI
Emotion	Einbindung von Komponenten in React
Figma	Erstellung von Mockups
Java & Spring	Backend-Entwicklung
MySQL	Datenbankmanagementsystem

Durch diese Auswahl an Technologien und Tools wurde sichergestellt, dass das System sowohl den funktionalen als auch den nicht-funktionalen Anforderungen gerecht wird und zukünftigen Erweiterungen oder Änderungen standhalten kann.