

Abschluss-Projekt Hotelbuchungssystem

Modul 223

Daniel Kovac und Kristian Lubina

The screenshot shows a hotel booking website for "Master Hotel". The top navigation bar includes links for "Check-in", "Check-out", "Rooms", "Booking", and "Guest Information". Below this, a large banner displays the text "Abschluss-Projekt Hotelbuchungssystem" and "Modul 223" along with the names "Daniel Kovac und Kristian Lubina". The main content area features two room options: "DZ KOMFORT" and "STANDARD DZ". Each room type has a small image, a description, and a price of "55,00 €". The "DZ KOMFORT" room includes amenities like a balcony and air conditioning. The "STANDARD DZ" room includes a balcony and heating.

Contents

Einleitung.....	6
Informieren und Planen	6
Anforderungsanalyse	6
User Story	6
Use-Case-Diagramm.....	6
ERM	7
Realisation des Back-Ends	8
Vorbereitungen für das Back-Ends.....	8
Projektanlage mit Spring Initializr	8
Datenbank einrichten und ins Projekt integrieren	9
JWT-Token ins Projekt integrieren.....	9
Dependencies implementieren.....	10
Grundaufbau des Backend realisieren.....	11
Entities und DTO.....	11
Erste Klassen realisieren	12
Booking-Klasse	12
Room-Klasse.....	13
DTO realisieren	14
UserDTO	14
RoomDTO	14
BookingDTO	14
Login realisieren	15
LoginRequest-Klasse.....	15
Response-Klasse.....	15
Repositories realisieren.....	16
BookingRepository	16
RoomRepository	16
UserRepository	16
CustomUserDetailsService realisieren.....	17
JWTUtils-Klasse realisieren	18
Security implementieren.....	19
CorsConfig realisieren.....	19
JWTAuthFilter realisieren	19
SecurityConfig realisieren.....	19
Zusammenarbeit der drei Klassen	20

AWS-Zugriffschlüssel erstellen	20
AwsS3Service-Klasse realisieren	21
Utils realisieren.....	22
Services realisieren.....	23
UserService realisieren.....	23
RoomService realisieren.....	24
BookingService realisieren.....	24
Controller realisieren	26
UserController realisieren.....	26
RoomController realisieren.....	27
BookingController realisieren.....	28
Backend testen	29
Testfall 1: User erstellen	29
Testfall 2: User einloggen.....	30
Testfall 3: Alle User abrufen	30
Testfall 4: Bookings testen	31
Testfall 5: User löschen	31
Fehlerbehandlung beim löschen testen.....	31
Richtiges löschen testen.....	32
Testfall 6: Zimmer hinzufügen	33
Backend-Architektur.....	35
Technologische Grundlagen.....	35
Architekturelle Schichten	35
Controller-Schicht.....	35
Service-Schicht.....	35
Repository-Schicht.....	35
Sicherheitsarchitektur.....	35
JWT-basierte Authentifizierung.....	35
Sicherheitskomponenten	36
Externe Integrationen	36
AWS S3 Integration	36
Datenmodell	36
Transaktionale Beziehungen	36
Initialisierung des Projekts	38
Abhängigkeiten installieren	40
Erstellung der Verzeichnisstruktur im Frontend	40

Kernverzeichnisse und deren Funktionen	40
Zusätzliche Verzeichnisse.....	40
Zugriffskontrolle und API-Kommunikation im Frontend	41
Web-Pages realisieren	42
Navigationsleiste (Navbar.js) erstellen.....	42
Startseite (HomePage.js) erstellen	43
Routing- und Seitenstruktur (App.jsx) erstellen	44
immer-Suchfunktion (RoomSearch.js) erstellen	45
Suchergebnisseite (RoomResult.jsx) erstellen	46
Zimmeransichtseite (AllRoomsPage.jsx) erstellen.....	47
Buchungssuchseite (FindBookingPage.jsx) erstellen	48
Zimmerdetailsseite (RoomDetailsPage.jsx) erstellen.....	49
Anmeldeseite (LoginPage.jsx) erstellen.....	50
Registrierungsseite (RegisterPage.jsx) erstellen	51
Benutzerprofilseite (ProfilePage.jsx) erstellen	52
Profilbearbeitungsseite (EditProfilePage.jsx) erstellen	53
Administration-Dashboard-Seite (AdminPage.jsx) erstellen.....	54
Zimmermanagement-Seite (ManageRoomPage.jsx) erstellen	55
Buchungsverwaltungsseite (ManageBookingsPage.jsx) erstellen	56
Zimmerhinzufügungsseite (AddRoomPage.jsx) erstellen	57
Buchungsbearbeitungsseite (EditBookingPage.jsx) erstellen	58
Zimmerbearbeitungsseite (EditRoomPage.jsx) erstellen.....	59
Routing- und Seitenstruktur (App.jsx) verbessern.....	60
Endresultat.....	61
Frontend testen.....	63
Testfall 1: Registrierung eines neuen Benutzers	63
Validierungstests.....	63
Testergebnisse	63
Testfall 2: Hinzufügen eines neuen Zimmers als Admin	64
Frontend-Architektur	65
Architekturelle Schichten	65
Sicherheitskonzept des Frontend	65
Testprotokoll Backend und Frontend	67
Sicherheitskonzept.....	68
Authentifizierung & Autorisierung	68
Datensicherheit.....	68

Schutz vor Angriffen	68
API-Sicherheit.....	68
Frontend-Sicherheit.....	68
Abläufe beim Login.....	69
Arbeitsjournal	70
Daniel Kovac.....	70
Kristian Lubina.....	73

Einleitung

Das Projekt umfasst die Entwicklung Hotelbuchungssystems, das mit Spring Boot im Backend und React im Frontend realisiert wurde.

Das System ermöglicht Benutzern die Online-Buchung von Hotelzimmern sowie Administratoren die effiziente Verwaltung von Zimmern, Buchungen und Benutzern.

Durch den Einsatz von JWT-basierter Authentifizierung, einer MySQL-Datenbank und AWS S3 für die Bildspeicherung bietet das System eine sichere und skalierbare Lösung für die digitale Hotelverwaltung.

Informieren und Planen

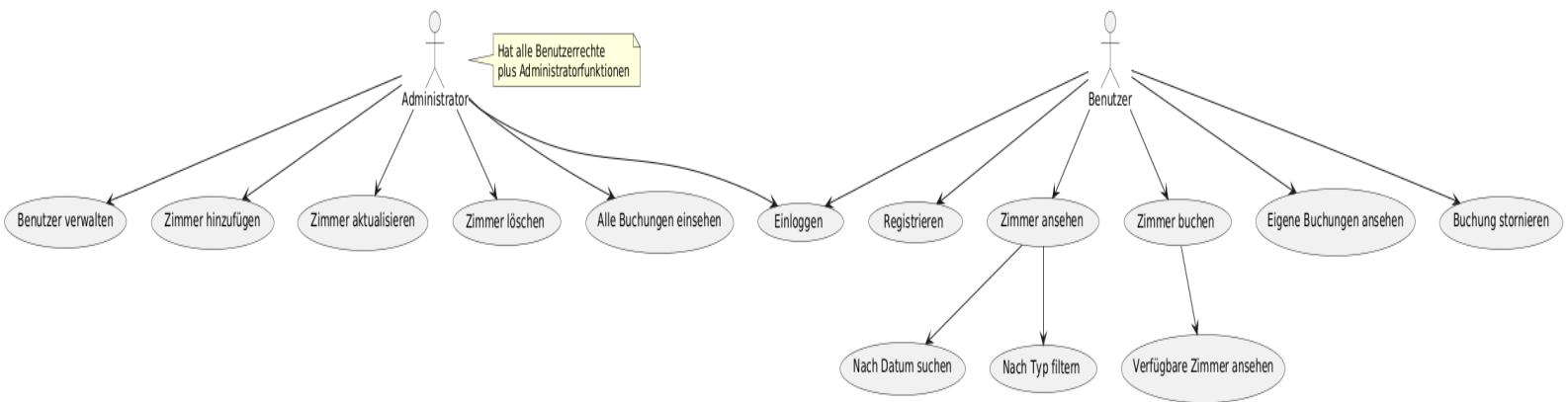
Anforderungsanalyse

User Story

Als potenzieller Hotelgast möchte ich über eine Webplattform ein Hotelzimmer nach meinen zeitlichen und preislichen Vorstellungen suchen, buchen und verwalten können, während Hoteladministratoren die Möglichkeit haben, das Zimmerangebot und die Buchungen effizient zu verwalten.

Use-Case-Diagramm

Das Use-Case-Diagramm veranschaulicht die wesentlichen Funktionen des Hotelbuchungssystems.



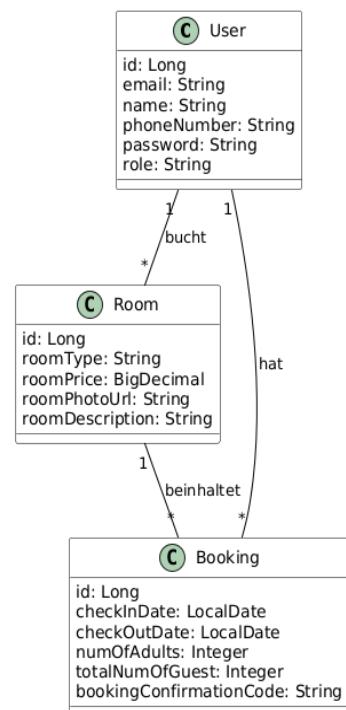
ERM

Das ERM bildet die grundlegende Datenbankstruktur eines Hotelbuchungssystems ab.

Die Hauptentitäten sind User (Benutzer), Room (Zimmer) und Booking (Buchung).

Benutzer können mehrere Zimmerbuchungen vornehmen, während ein Zimmer mehrere zeitlich getrennte Buchungen haben kann.

Jede Buchung speichert dabei wesentliche Informationen wie Check-in/Check-out Daten, Gästzahl und einen eindeutigen Bestätigungscode.



Realisation des Back-Ends

Vorbereitungen für das Back-Ends

Projektanlage mit Spring Initializr

The screenshot shows the Spring Initializr web application at start.spring.io. The left sidebar shows project settings: Project (Maven), Language (Java), and Spring Boot version (3.4.1). The right sidebar shows dependencies: Spring Web (selected), Spring Security, Spring Data JPA, MySQL Driver, Lombok, and Validation. Below the sidebar is a summary of Java version (23) and a 'GENERATE' button.

Als erstes haben wir das Projekt erstellt mit Hilfe von Spring Initializr. Die Konfiguration schafft eine Basis für ein sicheres Web-Projekt mit REST-API, Datenbankanbindung und einer effizienten Code-Basis.

Hierfür haben wir uns folgende Einstellung und Abhängigkeit entschieden.

Einstellungen

- **Build-Tool: Maven**
- **Sprache: Java**
- **Spring Boot Version: 3.4.1**
- **Metadaten:**
 - **Group:** com.dksynergy, **Artifact:** DKSynergy, **Name:** DKSynergy, **Beschreibung:** Projekt Modul 223.
 - **Package:** com.dksynergy.DKSynergy.
- **Packaging: Jar**
- **Java-Version: 23**

Abhängigkeiten

- **Spring Web:** REST-APIs und Web-Entwicklung (inkl. Tomcat).
- **Spring Security:** Authentifizierung und Autorisierung.
- **Spring Data JPA:** Datenbankzugriffe via Hibernate.
- **MySQL Driver:** Anbindung an MySQL-Datenbanken.
- **Lombok:** Reduziert Boilerplate-Code.
- **Validation:** Datenviadierung (z. B. @NotNull).

Datenbank einrichten und ins Projekt integrieren

The screenshot shows the MySQL Workbench interface. In the top-left pane, under 'SCHEMAS', there is a list with 'modul223' highlighted. The main pane is titled 'Query 1' and contains the SQL command: 'create database modul223;'. Below the query, the 'Output' pane shows a single row of results: '1 17:10:02 create database modul223'.

Die Datenbank "modul223" haben wir erfolgreich in MySQL erstellt und dies dient als Grundlage für die Speicherung aller relevanten Daten.

Nach der erfolgreichen Erstellung der Projekt-Datenbank haben wir die Datenbank auch in das Projekt eingebunden.

The screenshot shows the IntelliJ IDEA interface with a project named 'DKSynergy'. On the left, the project structure shows a 'src' folder containing 'main' and 'resources' subfolders. On the right, the 'application.properties' file is open, displaying the following configuration:

```

spring.application.name=DKSynergy
server.port=4040
spring.datasource.url=jdbc:mysql://localhost:3306/modul223
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update

```

JWT-Token ins Projekt integrieren

Um in unserer Anwendung JWT-basierte Authentifizierung zu integrieren, verwenden wir die JJWT-Bibliothek (Java JSON Web Token).

Die verschiedenen Module erfüllen spezifische Aufgaben:

1. **jjwt-impl**
 - Die Hauptbibliothek, die für das Erstellen, Signieren und Verifizieren von JWTs benötigt wird.
2. **jjwt-api**
 - Stellt die Schnittstelle bereit, über die JWT-Funktionen in der Anwendung aufgerufen werden können.
3. **jjwt-jackson**

- Ermöglicht die Nutzung von Jackson, um JSON-Daten einfach in den JWT-Body einzubetten oder auszulesen.

The screenshot shows three entries from the Maven Central Repository:

- 2. JWT :: Impl**: io.jsonwebtoken » [jjwt-impl](#). Last Release on Jun 21, 2024. 720 usages. Apache.
- 3. JWT :: API**: io.jsonwebtoken » [jjwt-api](#). Last Release on Jun 21, 2024. 688 usages. Apache.
- 4. JWT :: Extensions :: Jackson**: io.jsonwebtoken » [jjwt-jackson](#). Last Release on Jun 21, 2024. 630 usages. Apache.

Dependencies implementieren

Die benötigten Dependencies wurden in der pom.xml konfiguriert, darunter Spring Security für die Authentifizierung, sowie die JWT (JSON Web Token) Bibliotheken für eine sichere, tokenbasierte Benutzerauthentifizierung im System.

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-jackson</artifactId>
      <version>0.12.6</version>
      <scope>runtime</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-impl</artifactId>
      <version>0.12.6</version>
      <scope>runtime</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-api</artifactId>
    </dependency>
  </dependencies>

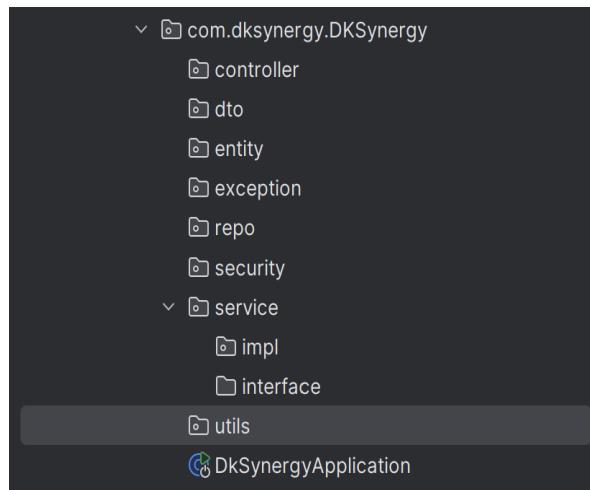
```

Grundaufbau des Backend realisieren

Die Backend-Struktur folgt dem Spring-Boot-Standard mit klar getrennten Packages:

- **controller:** Verarbeitung von HTTP-Anfragen und REST-Endpunkte
 - **dto:** Datentransfer-Objekte für sichere Datenübermittlung
 - **entity:** Datenbankmodelle für JPA/Hibernate
 - **exception:** Benutzerdefinierte Fehlerbehandlung
 - **repo:** Datenbankzugriff via Spring Data JPA
 - **security:** Authentifizierung und Autorisierung mit JWT
 - **service:** Geschäftslogik (mit impl/interface)
 - **utils:** Hilfsklassen und -methoden

```
graph TD; root["DkSynerg"] --> dto["dto"]; root --> entity["entity"]; root --> exception["exception"]; root --> repo["repo"]; root --> security["security"]; root --> service["service"]; service --> impl["impl"]; service --> interface["interface"]; root --> utils["utils"]
```



Diese modulare Architektur ermöglicht eine saubere Trennung der Verantwortlichkeiten und erleichtert Wartung sowie Erweiterungen.

```
.   _----_
 / \ / ___'_-__-_-___( )_---__- - - \ \ \ \
( ()\__| '_|| '_|| '_`\ \ \`| \ \ \ \
\ \/_ __) | \_)| \| \| \| \_)| \_) \_) )
 '  |_____| .__|_|_|_|_\__, | / / /
=====|_|=====|_|=/\_/_/_/
```

Nach erfolgreicher Initialisierung des Spring Boot Projekts (Version 3.4.1) und korrektem Start des Servers können wir mit der Implementierung der Entitäten und DTOs (Data Transfer Objects) für unser Hotelbuchungssystem fortfahren.

Entities und DTO

Die Klassen werden mit JPA-Annotationen konfiguriert:

- **@Entity**: Markiert die Klasse als JPA-Entity und damit als Datenbanktabelle.
 - **@Table(name = "users")**: Definiert den konkreten Tabellennamen in der Datenbank.
 - **@Data (Lombok)**: Generiert automatisch Getter, Setter, `toString()`, `equals()` und `hashCode()`-Methoden.

Die Felder werden wie folgt konfiguriert:

1. Primärschlüssel

- `@Id`: Markiert das Feld als Primärschlüssel
- `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Automatische ID-Generierung

2. Validierung und Constraints

- `@NotBlank`: Verhindert leere Pflichtfelder
- `@Column(unique = true)`: Stellt Einzigartigkeit sicher, z.B. für E-Mail-Adressen

3. Hauptfelder

- `email`: Eindeutige E-Mail-Adresse
- `name`: Benutzername
- `phoneNumber`: Telefonnummer
- `password`: Benutzerpasswort (wird verschlüsselt gespeichert)
- `role`: Benutzerrolle für Berechtigungen

4. Beziehungen

- `List<Booking> bookings`: Speichert die Buchungen eines Benutzers
- Wird später mit `@OneToMany` oder ähnlichen Relationen implementiert

Erste Klassen realisieren

Booking-Klasse

```

8
9      import java.util.ArrayList;
10     import java.util.List;
11
12     @Data no usages
13     @Entity
14     @Table(name = "users")
15     public class User {
16
17         @Id
18         @GeneratedValue(strategy = GenerationType.IDENTITY)
19         private long id;
20
21         @NotBlank(message = "E-Mail ist Notwendig")
22         @Column(unique = true)
23         private String email;
24         @NotBlank(message = "Name ist Notwendig")
25         private String name;
26         @NotBlank(message = "Telefonnummer ist Notwendig")
27         private String phoneNumber;
28         private String password;
29         private String role;
30         private List<Booking> bookings = new ArrayList<>();

```

Hier haben wir unsere **Booking-Klasse**, in der wir alle wichtigen Infos für eine Hotelbuchung oder Reservierung zusammenfassen.

Über die **JPA-Annotationen** (`@Entity`, `@Table` etc.) wird die Klasse direkt mit unserer Datenbank verknüpft, sodass die Daten in der Tabelle „bookings“ landen.

Damit das Ganze robust bleibt, setzen wir auf **Bean-Validierungen** (@NotNull, @Future, @Min). Die sorgen zum Beispiel dafür, dass keine negativen Werte für Kinder eingetragen werden können und das Check-out-Datum auch wirklich in der Zukunft liegt.

```

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;

@NotNull(message = "Bitte vervollständigen Sie Ihre Eincheck-Informationen.")
private LocalDate checkinDate;

@Future(message = "Das Check-out-Datum muss in der Zukunft liegen.")
private LocalDate checkoutDate;

@Min(value = 1, message = "Mindestens ein Erwachsener ist erforderlich.")
private int numOfAdults;

@Min(value = 0, message = "Es dürfen keine negativen Werte für Kinder angegeben werden.")
private int numOfChildren;

private int totalNumOfGuests;

private String bookingConfirmationCode;

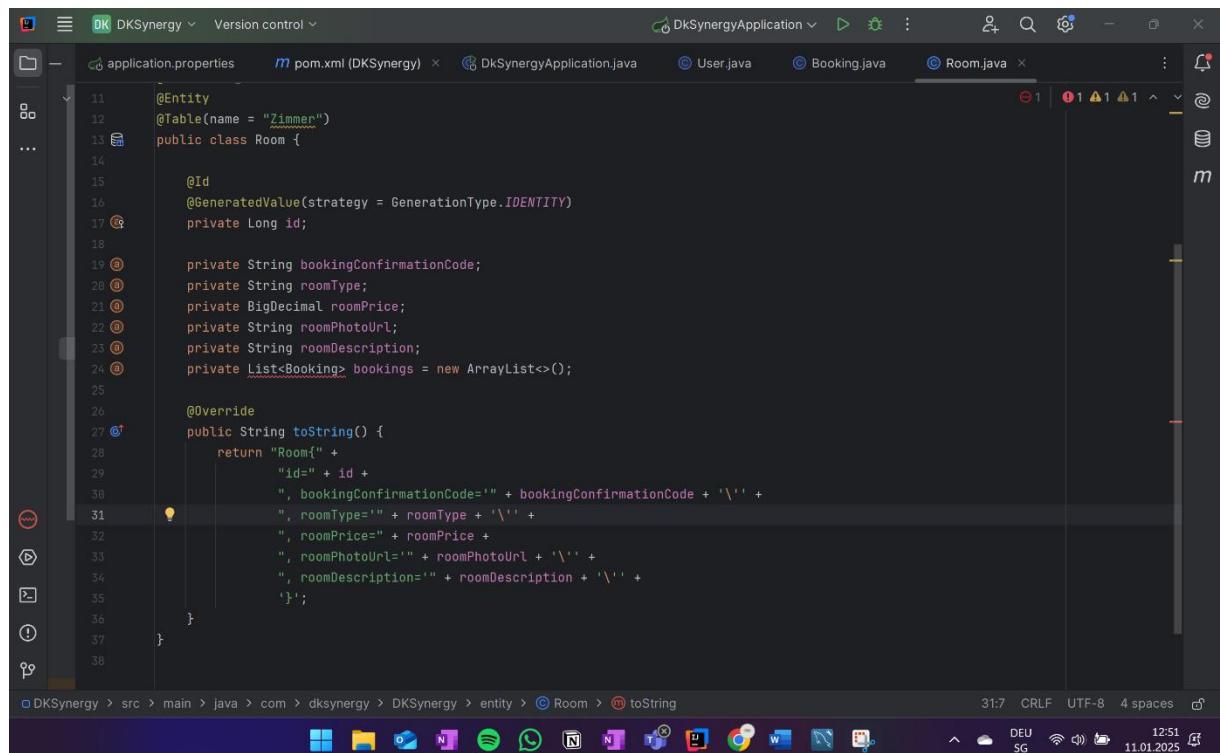
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "user_id")
private User user;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "room_id")

```

Zusätzlich haben wir noch ein Feld für den **Buchungsbestätigungscode** und Referenzen zu unseren **User**- und **Room**-Entitäten. Über die entsprechenden **Fetch-Typen** (EAGER oder LAZY) regeln wir, ob die verknüpften Daten sofort oder erst bei Bedarf geladen werden.

Room-Klasse



```

@Entity
@Table(name = "Zimmer")
public class Room {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String bookingConfirmationCode;
    private String roomType;
    private BigDecimal roomPrice;
    private String roomPhotoUrl;
    private String roomDescription;
    private List<Booking> bookings = new ArrayList<>();

    @Override
    public String toString() {
        return "Room{" +
            "id=" + id +
            ", bookingConfirmationCode='" + bookingConfirmationCode + '\'' +
            ", roomType='" + roomType + '\'' +
            ", roomPrice=" + roomPrice +
            ", roomPhotoUrl='" + roomPhotoUrl + '\'' +
            ", roomDescription='" + roomDescription + '\'' +
            '}';
    }
}

```

Hier haben wir unsere **Room-Klasse**, mit der wir verschiedene Zimmer in unserem System verwalten.

Über **@Entity** und **@Table(name = "Zimmer")** verknüpfen wir diese Klasse direkt mit der Datenbanktabelle „Zimmer“. Im Anschluss definieren wir Felder wie roomType, roomPrice und eine mögliche roomPhotoUrl, um unterschiedliche Zimmer und deren Eigenschaften abzubilden.

Mithilfe des **List bookings**-Felds möchten wir später mehrere Buchungen einem Zimmer zuordnen, was uns erlaubt, ganz einfach alle Buchungen pro Zimmer nachzuvollziehen. Über **Lombok (@Data)** bekommen wir außerdem wieder bequem unsere Getter, Setter und vieles mehr.

Die **toString()**-Methode rundet das Ganze ab und hilft uns, beim Debuggen oder Loggen schnell alle Infos zum Zimmer auf einen Blick zu sehen.

DTO realisieren

UserDTO

```
package com.dksynergy.DKSynergy.dto;

import com.dksynergy.DKSynergy.entity.Booking;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

import java.util.ArrayList;
import java.util.List;

@Data no usages
@JsonInclude(JsonInclude.Include.NON_NULL)
public class UserDTO {

    private Long id;
    private String email;
    private String name;
    private String phoneNumber;
    private String role;
    private List<Booking> bookings = new ArrayList<>();
}
```

Mit unserer **UserDTO** können wir die Benutzerdaten wie id, email, name und so weiter ganz einfach weiterreichen, zum Beispiel wenn wir Daten an ein Frontend schicken.

Die **@JsonInclude**-Annotation sorgt dafür, dass Felder mit null im JSON ausgeblendet werden und so die Antwort schlanker ausfällt.

Außerdem haben wir hier noch eine Liste von **Bookings**, damit wir auf Wunsch gleich sehen, welche Buchungen ein User hat. Dank **@Data** (von Lombok) müssen wir uns um Getter, Setter & Co. nicht extra kümmern.

RoomDTO

Mit **RoomDTO** können wir wichtige Zimmergegebenheiten wie Typ, Preis und Beschreibung abbilden.

BookingDTO

BookingDTO abbildet die verschiedenen Buchungsdetails (Check-in, Check-out, Gästezahlen etc.).

Login realisieren

LoginRequest-Klasse

```

1 package com.dksynergy.DKSynergy.dto;
2
3 import jakarta.validation.constraints.NotBlank;
4 import lombok.Data;
5
6 @Data no usages
7 public class LoginRequest {
8
9     @NotBlank(message = "E-Mail-Adresse ist erforderlich.")
10    private String email;
11    @NotBlank(message = "Passwort ist erforderlich.")
12    private String password;
13
14 }
15

```

Wir haben diese Klasse als nächsten Schritt eingeführt, um die Login-Daten (E-Mail und Passwort) klar vom restlichen Code zu trennen.

Response-Klasse

Wir haben die **Response-Klasse** erweitert, um flexibler auf unterschiedliche Anforderungen reagieren zu können.

Neben grundlegenden Feldern wie statusCode, message und token gibt es jetzt auch Platz für einzelne Objekte wie UserDTO, RoomDTO oder BookingDTO.

Zusätzlich haben wir die Möglichkeit geschaffen, Listen von Benutzern, Zimmern oder Buchungen zurückzugeben (userList, roomList, bookingList).

```

1 package com.dksynergy.DKSynergy.dto;
2
3 import com.fasterxml.jackson.annotation.JsonInclude;
4 import lombok.Data;
5
6 import java.util.List;
7
8 @Data no usages
9 @JsonInclude(JsonInclude.Include.NON_NULL)
10 public class Response {
11
12     private int statusCode;
13
14     private String message;
15     private String token;
16     private String role;
17     private String expirationTime;
18     private String bookingConfirmationCode;
19
20     private UserDTO user;
21     private RoomDTO room;
22     private BookingDTO booking;
23     private List<UserDTO> userList;
24     private List<RoomDTO> roomList;
25     private List<BookingDTO> bookingList;
26
27 }
28

```

Repositories realisieren

BookingRepository

```

package com.dksynergy.DKSynergy.repo;

import com.dksynergy.DKSynergy.entity.Booking;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface BookingRepository extends JpaRepository<Booking, Long> {
    List<Booking> findByBookingId (Long roomId);
    List<Booking> findByBookingConfirmationCode(String confirmationCode);
    List<Booking> findByUserId (Long userId);
}

```

Hier haben wir die Methoden für unsere **Booking-Entität** definiert, um spezifische Abfragen zu ermöglichen:

- **findByBookingId**: Um Buchungen anhand der Zimmer-ID zu suchen.
- **findByBookingConfirmationCode**: Zum Abrufen einer Buchung über den Bestätigungscode.
- **findById**: Für alle Buchungen eines bestimmten Nutzers.

RoomRepository

Dieses Repository erweitert die Funktionalität der **Room-Entität**. Es ermöglicht:

- **findDistinctRoomTypes**: Um die verschiedenen Raumtypen zu ermitteln.
- **findAvailableRoomsByDatesAndTypes**: Hiermit können freie Zimmer für bestimmte Daten und Typen abgefragt werden.
- **getAllAvailableRooms**: Um alle verfügbaren Zimmer zu bekommen, die momentan nicht gebucht sind.

UserRepository

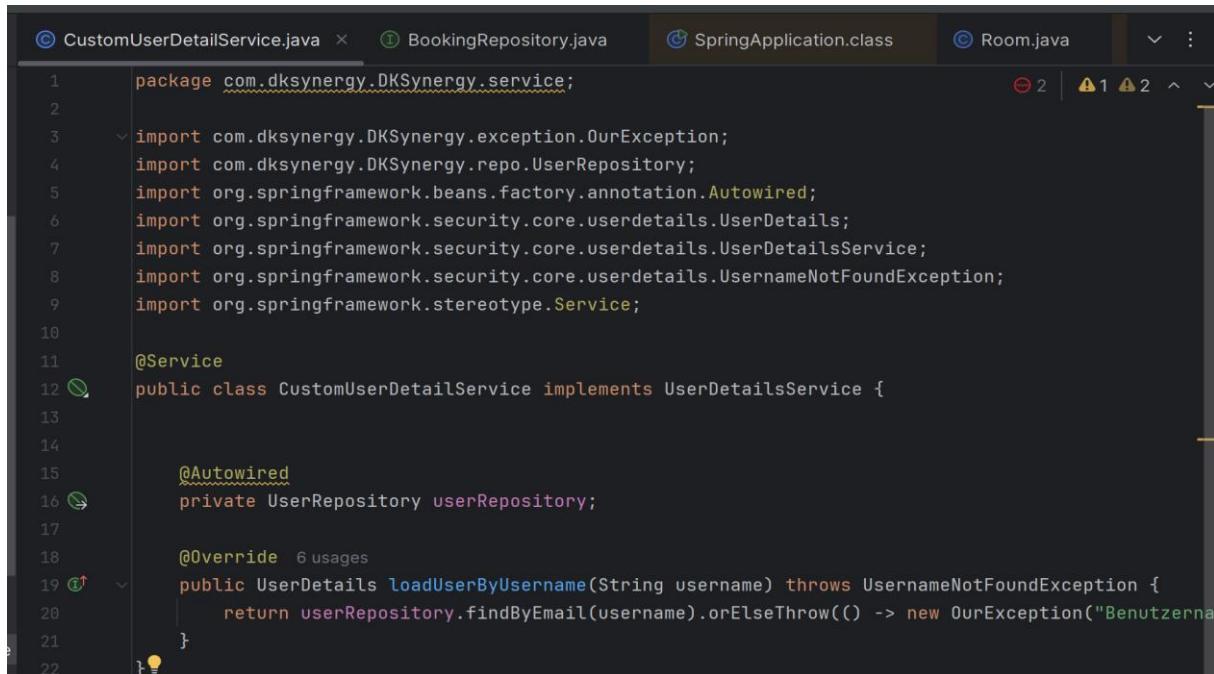
Dieses Repository ergänzt die **User-Entität** mit speziellen Abfragemethoden. Es ermöglicht:

- **existsByEmail**: Prüft, ob ein Nutzer mit einer bestimmten E-Mail-Adresse existiert.
- **findByEmail**: Sucht Nutzer anhand ihrer E-Mail-Adresse.

CustomUserDetailsService realisieren

Um Benutzerinformationen aus unserer Datenbank zu laden, wenn ein Nutzer sich anmeldet, haben wir einen **CustomUserDetailsService** eingerichtet, der die **UserDetailsService**-Schnittstelle von Spring Security implementiert.

Mit dieser Implementierung können wir unsere Benutzerverwaltung flexibel gestalten und an die spezifischen Anforderungen unseres Systems anpassen. Außerdem ist die Nutzung des Repositories ein klarer und einfacher Weg, um die Datenbank anzubinden.



The screenshot shows a Java code editor with the following code:

```
1 package com.dksynergy.DKSynergy.service;
2
3 import com.dksynergy.DKSynergy.exception.OurException;
4 import com.dksynergy.DKSynergy.repo.UserRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.security.core.userdetails.UserDetails;
7 import org.springframework.security.core.userdetails.UserDetailsService;
8 import org.springframework.security.core.userdetails.UsernameNotFoundException;
9 import org.springframework.stereotype.Service;
10
11 @Service
12 public class CustomUserDetailsService implements UserDetailsService {
13
14
15     @Autowired
16     private UserRepository userRepository;
17
18     @Override 6 usages
19     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
20         return userRepository.findByEmail(username).orElseThrow(() -> new OurException("Benutzername nicht gefunden"));
21     }
22 }
```

JWTUtils-Klasse realisieren

Wir haben die **JWTUtils**-Klasse erstellt, die sich um das Generieren, Validieren und Analysieren von JWT-Token kümmert.

The screenshot shows a Windows PowerShell window at the top with the following command and output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Installieren Sie die neueste PowerShell für neue Funktionen und Verbesserungen! https://aka.ms/PSWindows

PS C:\Users\Daniel Kovac> [convert]::ToString((1..32 | ForEach-Object {Get-Random -Minimum 0 -Maximum 256}))v/8Y0yXkEbouzg0MvyVq4A2efym2LW5uyuFbCH+rdj8=
PS C:\Users\Daniel Kovac> |
```

Below it is a screenshot of the VS Code IDE. The left sidebar shows a project structure for 'DKSynergy' with several Java files like 'CustomUserDetailService.java', 'CorsConfig.java', 'JWTAuthFilter.java', and 'JWTUtils.java'. The right pane displays the code for 'JWTUtils.java'.

```
import java.util.Base64;
import java.util.Date;
import java.util.function.Function;

@Service no usages
public class JWTUtils {

    private static final long EXPIRATION_TIME = 1000 * 60 * 24 * 7; //Für 1 Woche 1 usage

    private final SecretKey key; 3 usages

    public JWTUtils() {
        String secretString = "v/8Y0yXkEbouzg0MvyVq4A2efym2LW5uyuFbCH+rdj8=";
        byte[] keyBytes = Base64.getDecoder().decode(secretString.getBytes(StandardCharsets.UTF_8));
        this.key = new SecretKeySpec(keyBytes, algorithm: "HmacSHA256");
    }

    public String generateToken(UserDetails userDetails) { no usages
        return Jwts.builder()
            .subject(userDetails.getUsername())
            .issuedAt(new Date(System.currentTimeMillis()))
            .expiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
            .signWith(key)
            .compact();
    }
}
```

The status bar at the bottom of the IDE shows the file path 'DKSynergy > src > main > java > com > dksynergy > DKSynergy > utils > JWTUtils > JWTUtils.java', the encoding 'UTF-8', and the date '25.01.2025'.

Die **JWTUtils** hat folgende **Funktionen**:

- 1. Token-Generierung (generateToken)**

Diese Methode erstellt einen neuen JWT für den Benutzer basierend auf seinem username.

- 2. Username extrahieren (extractUsername)**

Diese Methode zieht den Benutzernamen (oder die E-Mail) aus dem Token heraus.

- 3. Token-Validierung (isValidToken)**

Diese Funktion überprüft, ob das Token noch gültig ist.

- 4. Token-Ablauf überprüfen (isTokenExpired)**

Diese Methode überprüft, ob das Token basierend auf seiner Ablaufzeit noch gültig ist.

- 5. Schlüsselmanagement**

In der Konstruktor-Methode (JWTUtils) wird ein SecretKey generiert, der für die Signatur der JWT verwendet wird.

Mit dieser Klasse schaffen wir die Grundlage für eine sichere und skalierbare JWT-basierte Authentifizierung.

Security implementieren

CorsConfig realisieren

Die **CorsConfig**-Klasse dient der Konfiguration von **Cross-Origin Resource Sharing (CORS)** in unserer Anwendung.

CORS ist notwendig, wenn unser Backend von einer anderen Domain, beispielsweise einem Frontend, aufgerufen wird.

In dieser Klasse definieren wir:

- **Globale Regeln:** Alle Endpunkte (/**) dürfen von beliebigen Ursprüngen (*) angesprochen werden, unabhängig von der Domain.
- **HTTP-Methoden:** Wir erlauben die Methoden GET, POST, PUT, DELETE, was typische API-Anfragen abdeckt.

JWTAuthFilter realisieren

Der **JWTAuthFilter** ist ein Filter, der bei jeder HTTP-Anfrage ausgeführt wird.

Seine Aufgabe ist es, die Sicherheit unserer Anwendung zu gewährleisten, indem er JWT-Tokens verarbeitet und überprüft.

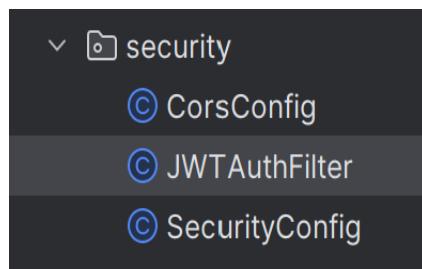
SecurityConfig realisieren

Die **SecurityConfig**-Klasse ist die zentrale Sicherheitskonfiguration der Anwendung.

Sie definiert:

1. **Routen und Zugriffskontrollen:**
 - Bestimmte Endpunkte wie /auth/**, /rooms/** und /bookings/** sind öffentlich zugänglich.
 - Alle anderen Endpunkte erfordern Authentifizierung.
2. **Stateless Session-Management:**
 - Sessions werden deaktiviert (STATELESS), da JWT-Tokens verwendet werden, um Benutzer zu authentifizieren.
3. **JWTAuthFilter einbinden:**
 - Der Filter wird so konfiguriert, dass er vor dem UsernamePasswordAuthenticationFilter ausgeführt wird. Dadurch können Tokens bereits frühzeitig geprüft werden.
4. **Authentifizierung und Passwortmanagement:**
 - Der **DaoAuthenticationProvider** wird verwendet, um Benutzer aus unserer Datenbank über den **CustomUserDetailsService** zu laden.
 - Mit dem **BCryptPasswordEncoder** stellen wir sicher, dass Passwörter sicher gespeichert und überprüft werden.
5. **Flexibilität durch Beans:**
 - AuthenticationManager: Verwaltet die Authentifizierung und wird in anderen Services genutzt.
 - PasswordEncoder: Kann überall im Code verwendet werden, um Passwörter zu verschlüsseln oder zu prüfen.

Zusammenarbeit der drei Klassen



- CorsConfig** ermöglicht den Zugriff auf unsere API von externen Quellen.
- JWTAuthFilter** überprüft bei jeder Anfrage die Authentifizierung durch JWT.
- SecurityConfig** orchestriert die gesamte Sicherheitsarchitektur und sorgt für klare Regeln und eine sichere Authentifizierung.

Mit dieser Konfiguration haben wir eine moderne, flexible und sichere Grundlage für eine RESTful API geschaffen, die auf JWT-Authentifizierung basiert.

AWS-Zugriffschlüssel erstellen

```

    spring.application.name=DkSynergy
    server.port=4040

    spring.datasource.url=jdbc:mysql://localhost:3306/modul223
    spring.datasource.username=root
    spring.datasource.password=root
    spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

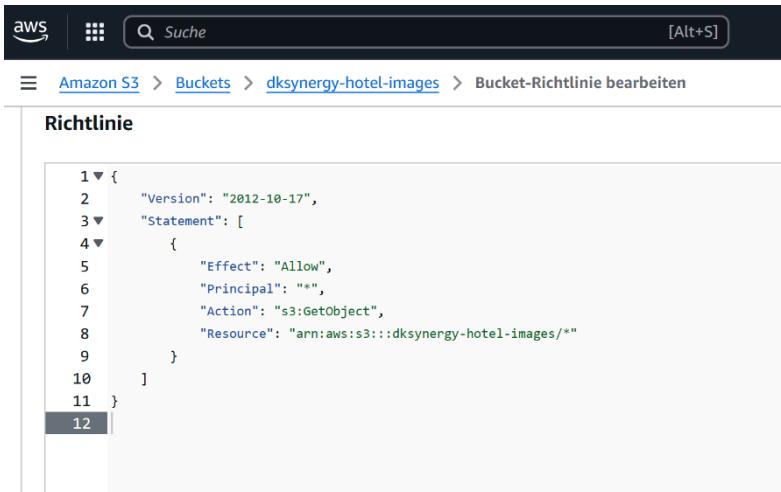
    spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
    spring.jpa.hibernate.ddl-auto=update

    aws.s3.secret.key=YJ7IsCAsZ1zH3fgKt7qtyVz+AyGlw/LM23jPM2m8
    aws.s3.access.key=AKIAU6GDYPAQPZLTKI
  
```

Wir haben einen **AWS-Zugriffschlüssel** erstellt, um die Verbindung unserer Anwendung mit AWS-Diensten zu ermöglichen. Dadurch können wir jetzt AWS-Ressourcen wie S3 oder Secrets Manager sicher ansprechen.

The screenshot shows the AWS IAM 'Create New Access Key' wizard. Step 1: 'Zugriffschlüssel erstellt' (Key created) - A note says: 'Dies ist das einzige Mal, dass der geheime Zugriffschlüssel angezeigt oder heruntergeladen werden kann. Sie können ihn später nicht wiederherstellen. Sie können jedoch jederzeit einen neuen Zugriffschlüssel erstellen.' Step 2: 'Zugriffschlüssel' - It lists the generated access key (AKIAU6GDYPAQPZLTKI) and its corresponding secret key (YJ7IsCAsZ1zH3fgKt7qtyVz+AyGlw/LM23jPM2m8). Step 3: 'Bewährte Methoden für Zugriffschlüssel' - It provides tips for secure key management.

Um die Schlüssel sicher zu verwalten, ist geplant, sie später in Umgebungsvariablen oder einem Secrets Manager abzulegen. Der Zugriff ist derzeit so konfiguriert, dass die Anwendung nahtlos mit den notwendigen Diensten arbeiten kann.



```

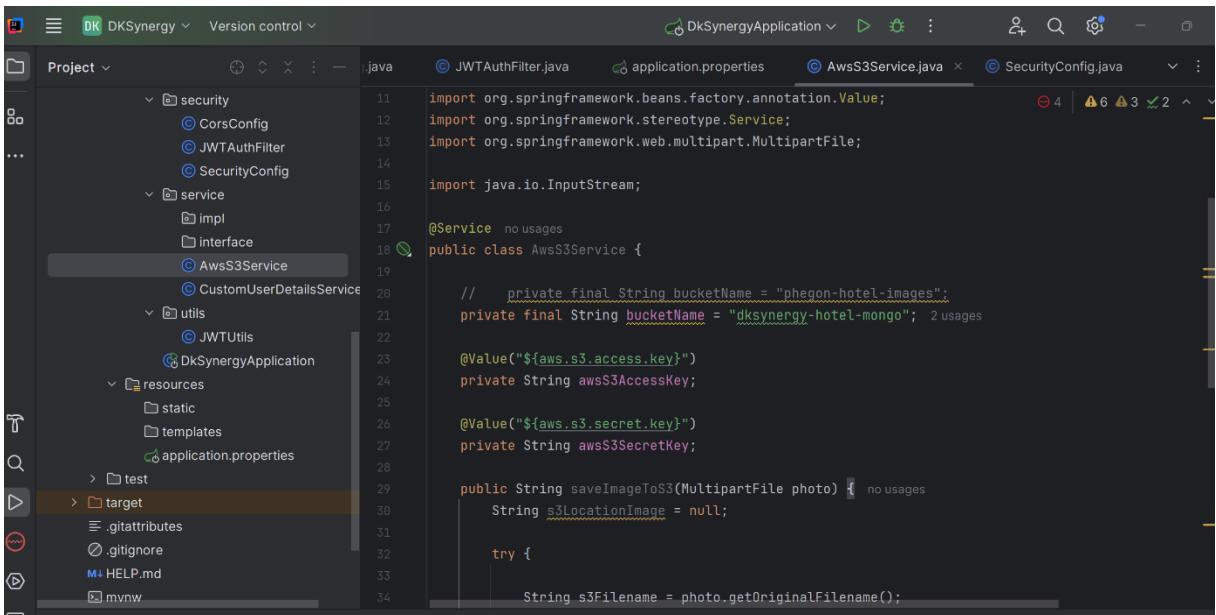
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Principal": "*",
7              "Action": "s3:GetObject",
8              "Resource": "arn:aws:s3:::dksynergy-hotel-images/*"
9          }
10     ]
11 }
12

```

Wir haben eine **Bucket-Policy** für den S3-Bucket dksynergy-hotel-images hinzugefügt. Diese Konfiguration ermöglicht es, Hotelbilder ohne zusätzliche Authentifizierung bereitzustellen. Wichtig ist, dass der Bucket nur für öffentliche Inhalte verwendet wird.

AwsS3Service-Klasse realisieren

Wir haben die Klasse **AwsS3Service** implementiert, um Bilder in einem S3-Bucket zu speichern.



```

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.InputStream;

@Service no usages
public class AwsS3Service {

    // private final String bucketName = "phegon-hotel-images";
    private final String bucketName = "dksynergy-hotel-mongo"; 2 usages

    @Value("${aws.s3.access.key}")
    private String awsS3AccessKey;

    @Value("${aws.s3.secret.key}")
    private String awsS3SecretKey;

    public String saveImageToS3(MultipartFile photo) { no usages
        String s3locationImage = null;

        try {

            String s3filename = photo.getOriginalFilename();

```

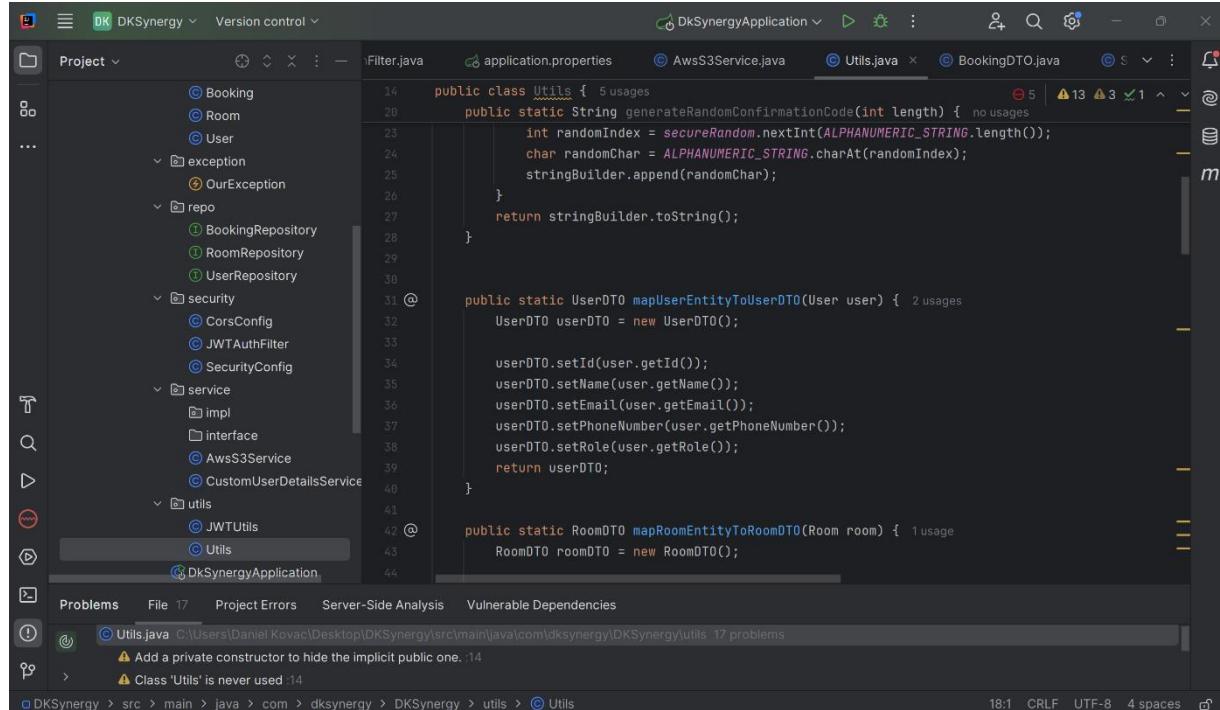
Die Methode `saveImageToS3(MultipartFile photo)` übernimmt die hochgeladene Datei und speichert sie in unserem S3-Bucket.

Dabei wird:

- Der **S3-Client** mithilfe von Zugriffsschlüsseln (Access Key und Secret Key) initialisiert.
- Die Datei mit einem bestimmten Dateinamen in den Bucket hochgeladen.
- Eine öffentlich zugängliche URL zur hochgeladenen Datei zurückgegeben.

Utils realisieren

Wir haben die **Utils-Klasse** erstellt, um häufig genutzte Funktionen und Mappings in der Anwendung zentral zu verwalten.



```

14  public class Utils { 5 usages
15
16      public static String generateRandomConfirmationCode(int length) { no usages
17          int randomIndex = secureRandom.nextInt(ALPHANUMERIC_STRING.length());
18          char randomChar = ALPHANUMERIC_STRING.charAt(randomIndex);
19          stringBuilder.append(randomChar);
20      }
21      return stringBuilder.toString();
22  }
23
24  @
25
26  public static UserDTO mapUserEntityToUserDTO(User user) { 2 usages
27      UserDTO userDTO = new UserDTO();
28
29      userDTO.setId(user.getId());
30      userDTO.setName(user.getName());
31      userDTO.setEmail(user.getEmail());
32      userDTO.setPhoneNumber(user.getPhoneNumber());
33      userDTO.setRole(user.getRole());
34      return userDTO;
35  }
36
37  @
38  public static RoomDTO mapRoomEntityToRoomDTO(Room room) { 1 usage
39      RoomDTO roomDTO = new RoomDTO();
40
41
42  }
43
44

```

The screenshot shows the Eclipse IDE interface with the 'Utils.java' file open in the central editor window. The code defines a static method 'generateRandomConfirmationCode' that generates a random string of a specified length. It also contains two static methods for mapping 'User' and 'Room' entities to their respective DTOs ('UserDTO' and 'RoomDTO'). The 'Utils' class is located in the 'com.dksynergy.DkSynergy.utils' package. The bottom status bar indicates the file is 17 lines long and has 17 problems.

Die Utils-Klasse bietet unter anderem folgende Funktionen:

- **Generierung von BestätigungsCodes**
- **Mapping von Entitäten zu DTOs**
- **Erweiterte Mapping-Methoden**
- **Listen-Mappings**

Mit der Utils-Klasse haben wir eine zentrale Komponente geschaffen, die die Verarbeitung und Darstellung von Daten vereinfacht und gleichzeitig die Wartbarkeit der Anwendung verbessert.

Services realisieren

UserService realisieren

Die **UserService**-Klasse implementiert alle Kernfunktionen, die sich auf Benutzer in unserer Anwendung beziehen, wie Registrierung, Login, Abrufen von Benutzerinformationen oder Löschen eines Nutzers.

Sie arbeitet als Brücke zwischen Repositorys, Sicherheitsmechanismen und Business-Logik, um die Daten zentral zu verarbeiten und kontrolliert bereitzustellen.

```

21  public class UserService implements IUserService {
22      @Autowired
23      private UserRepository userRepository;
24      @Autowired
25      private PasswordEncoder passwordEncoder;
26      @Autowired
27      private JWTUtils jwtUtils;
28      @Autowired
29      private AuthenticationManager authenticationManager;
30
31      @Override no usages
32      public Response register(User user) {
33          Response response = new Response();
34          try {
35              if (user.getRole() == null || user.getRole().isBlank()) {
36                  user.setRole("USER");
37              }
38              if (userRepository.existsByEmail(user.getEmail())) {
39                  throw new OurException(user.getEmail() + " existiert bereits");
40              }
41              user.setPassword(passwordEncoder.encode(user.getPassword()));
42              User savedUser = userRepository.save(user);
43              UserDTO userDTO = Utils.mapUserEntityToUserDTO(savedUser);
44
        
```

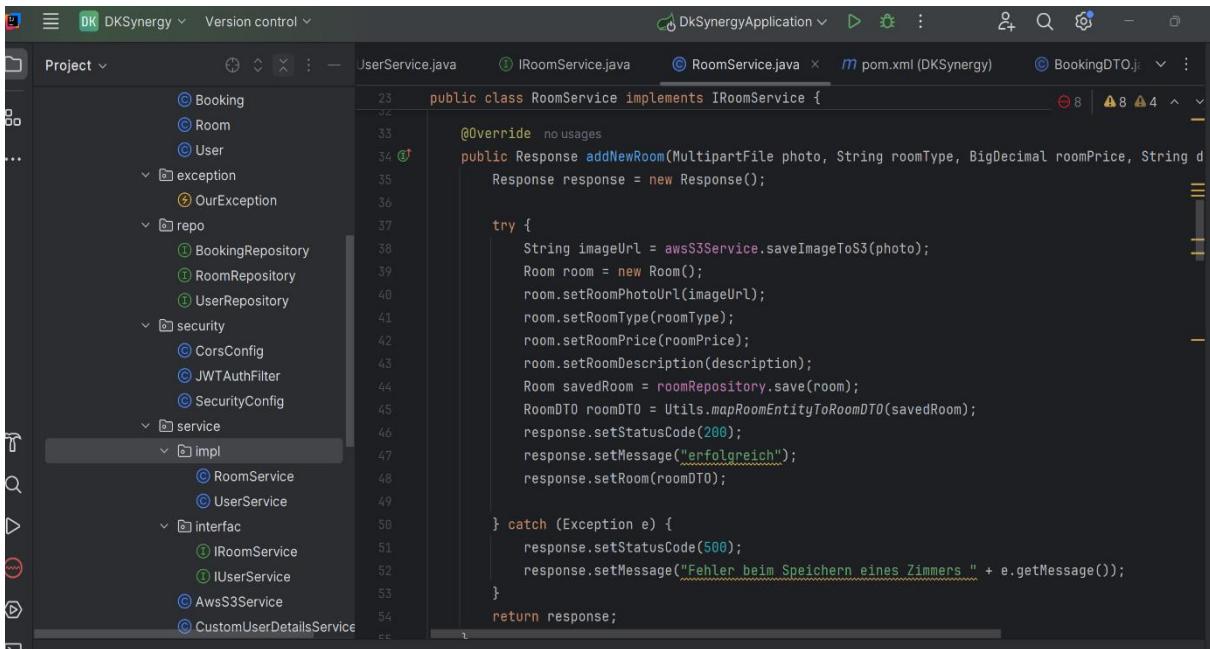
The screenshot shows the Eclipse IDE interface with the UserService.java file open. The code implements the IUserService interface and provides implementations for various methods. The code includes annotations like @Autowired for dependency injection and handles user registration, login, and retrieval. There are also comments and some placeholder code (like the empty @Override annotation). The bottom status bar shows the file path as C:\Users\Daniel Kovac\Desktop\DKSynergy\src\main\java\com\dksynergy\DKSynergy\service\impl\UserService.java and the current date and time as 25.01.2025.

Die Klasse beinhaltet folgende Methoden:

1. **register(User user)**
Registrierung eines neuen Benutzers.
2. **login(LoginRequest loginRequest)**
Authentifizierung eines Benutzers und Generierung eines JWT-Tokens.
3. **getAllUsers()**
Abrufen aller Benutzer in der Datenbank.
4. **getUserBookingHistory(String userId)**
5. **deleteUser(String userId)**
Löschen eines Benutzers anhand der ID.
6. **getUserById(String userId)**
Abrufen eines einzelnen Benutzers anhand seiner ID.
7. **getMyInfo(String email)**
Abrufen der Informationen des aktuell angemeldeten Benutzers anhand seiner E-Mail.

RoomService realisieren

Die Klasse RoomService implementiert das IRoomService-Interface und kapselt damit sämtliche Logik (Business Logic) rund um das Management von Zimmern.



```

public class RoomService implements IRoomService {
    @Override no usages
    public Response addNewRoom(MultipartFile photo, String roomType, BigDecimal roomPrice, String d
        Response response = new Response();

        try {
            String imageUrl = awsS3Service.saveImageToS3(photo);
            Room room = new Room();
            room.setRoomPhotoUrl(imageUrl);
            room.setRoomType(roomType);
            room.setRoomPrice(roomPrice);
            room.setRoomDescription(description);
            Room savedRoom = roomRepository.save(room);
            RoomDTO roomDTO = Utils.mapRoomEntityToRoomDTO(savedRoom);
            response.setStatusCode(200);
            response.setMessage("Erfolgreich");
            response.setRoom(roomDTO);

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setMessage("Fehler beim Speichern eines Zimmers " + e.getMessage());
        }
        return response;
}

```

Die Haupt-Funktionen der Klasse sind folgt:

- Zimmer erstellen
- Alle Zimmertypen abrufen
- Alle Zimmer abrufen
- Zimmer löschen
- Zimmer aktualisieren
- Zimmer nach ID suchen
- Verfügbare Zimmer im Zeitraum liefern
- Alle verfügbaren Zimmer liefern

BookingService realisieren

Die **RoomService**-Klasse implementiert die gesamte Logik rund um Zimmermanagement in unserer Anwendung.

Sie ermöglicht das Hinzufügen, Bearbeiten, Abrufen und Löschen von Zimmern sowie die Verwaltung von verfügbaren Zimmern basierend auf Buchungsdaten.

Ausserdem verbindet sie verschiedene Komponenten wie Repositorys und den AWS S3-Service für Bildspeicherung.

```

22  public class BookingService implements IBookingService {
23      private UserRepository userRepository;
24
25      @Override no usages
26      public Response saveBooking(Long roomId, Long userId, Booking bookingRequest) {
27
28          Response response = new Response();
29
30          try {
31              if (bookingRequest.getCheckoutDate().isBefore(bookingRequest.getCheckinDate())) {
32                  throw new IllegalArgumentException("Das Check-in-Datum muss nach dem Check-out-Datum sein");
33              }
34
35              Room room = roomRepository.findById(roomId).orElseThrow(() -> new OurException("Zimmer mit der ID " + roomId + " nicht gefunden"));
36
37              User user = userRepository.findById(userId).orElseThrow(() -> new OurException("Benutzer mit der ID " + userId + " nicht gefunden"));
38
39              List<Booking> existingBookings = room.getBookings();
40
41              if (!roomIsAvailable(bookingRequest, existingBookings)) {
42                  throw new OurException("Zimmer für den ausgewählten Datumsbereich nicht verfügbar");
43              }
44
45              bookingRequest.setRoom(room);
46              bookingRequest.setUser(user);
47              String bookingConfirmationCode = Utils.generateRandomConfirmationCode(length: 10);
48              bookingRequest.setBookingConfirmationCode(bookingConfirmationCode);
49              bookingRepository.save(bookingRequest);
50
51              response.setStatusCode(200);
52
53          } catch (OurException e) {
54              response.setStatusCode(e.getStatusCode());
55              response.setErrorMessage(e.getMessage());
56
57          }
58
59      }
60
61      private boolean roomIsAvailable(BookingRequest bookingRequest, List<Booking> existingBookings) {
62          for (Booking existingBooking : existingBookings) {
63              if (existingBooking.getCheckinDate().isAfter(bookingRequest.getCheckinDate()) && existingBooking.getCheckoutDate().isBefore(bookingRequest.getCheckoutDate()))
64                  return false;
65          }
66
67          return true;
68      }
69
70  }

```

Hierfür werden folgende Methoden genutzt:

- 1. addNewRoom**
Fügt ein neues Zimmer hinzu.
- 2. getAllRoomTypes**
Gibt eine Liste aller verfügbaren Raumtypen zurück.
- 3. getAllRooms**
Gibt eine Liste aller Zimmer zurück.
- 4. deleteRoom**
Löscht ein Zimmer anhand seiner ID.
- 5. updateRoom**
Aktualisiert die Details eines Zimmers.
- 6. getRoomById**
Gibt ein einzelnes Zimmer anhand seiner ID zurück.
- 7. getAvailableRoomsByDataAndType**
Gibt eine Liste von verfügbaren Zimmern basierend auf Check-in- und Check-out-Datum sowie dem Zimmertyp zurück.
- 8. getAllAvailableRooms**
Gibt alle derzeit verfügbaren Zimmer zurück.

Controller realisieren

UserController realisieren

Die **UserController**-Klasse dient als Schnittstelle zwischen der Anwendung und den Benutzerbezogenen API-Endpunkten.

Sie definiert, wie externe Anfragen für Benutzeroperationen wie Abrufen von Benutzerinformationen, Löschen von Benutzern oder das Anzeigen von Buchungshistorien verarbeitet werden.

```

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private IUserService userService;

    @GetMapping("/all")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> getAllUsers() {
        Response response = userService.getAllUsers();
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }

    @GetMapping("/get-by-id/{userId}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> getUserId(@PathVariable("userId") String userId) {
        Response response = userService.getUserId(userId);
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }

    @DeleteMapping("/delete/{userId}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> deleteUser(@PathVariable("userId") String userId) {
        Response response = userService.deleteUser(userId);
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }
}

```

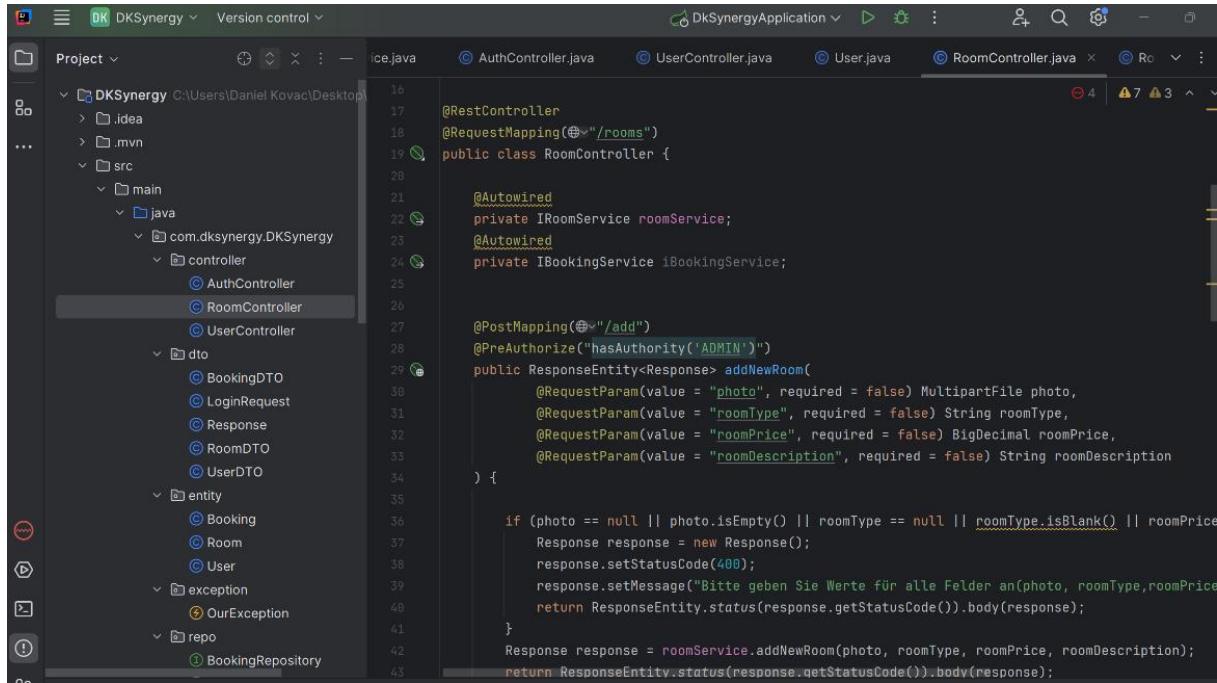
Zur Realisierung werden folgende Methoden genutzt:

- 1. getAllUsers**
Gibt alle Benutzer der Datenbank zurück.
- 2. getUserId**
Gibt die Informationen eines einzelnen Benutzers anhand seiner ID zurück.
- 3. deleteUser**
Löscht einen Benutzer aus der Datenbank.
- 4. getLoggedInUserProfile**
Gibt die Informationen des aktuell angemeldeten Benutzers zurück.
- 5. getUserBookingHistory**
Gibt die Buchungshistorie eines Benutzers zurück.

RoomController realisieren

Die **RoomController**-Klasse definiert alle Endpunkte, die für die Verwaltung von Zimmern erforderlich sind.

Sie fungiert als Schnittstelle zwischen der Anwendung und den HTTP-Anfragen für das Zimmermanagement, wie das Hinzufügen, Bearbeiten, Abrufen oder Löschen von Zimmern sowie das Anzeigen von Verfügbarkeiten.



The screenshot shows the IntelliJ IDEA interface with the project 'DKSynergy' open. The left sidebar displays the project structure under 'src/main/java/com.dksynergy.DKSynergy/controller'. The 'RoomController' class is selected and shown in the main editor window. The code implements a REST controller for rooms, using annotations like @RestController and @RequestMapping. It includes methods for adding a new room, getting all rooms, getting room types, getting available rooms by date and type, updating a room, and deleting a room. The code also handles validation and response generation.

```

16  @RestController
17  @RequestMapping(value = "/rooms")
18  public class RoomController {
19      @Autowired
20      private IRoomService roomService;
21      @Autowired
22      private IBookingService iBookingService;
23
24      @PostMapping(value = "/add")
25      @PreAuthorize("hasAuthority('ADMIN')")
26      public ResponseEntity<Response> addNewRoom(
27          @RequestParam(value = "photo", required = false) MultipartFile photo,
28          @RequestParam(value = "roomType", required = false) String roomType,
29          @RequestParam(value = "roomPrice", required = false) BigDecimal roomPrice,
30          @RequestParam(value = "roomDescription", required = false) String roomDescription
31      ) {
32
33          if (photo == null || photo.isEmpty() || roomType == null || roomType.isBlank() || roomPrice == null) {
34              Response response = new Response();
35              response.setStatusCode(400);
36              response.setMessage("Bitte geben Sie Werte für alle Felder an(photo, roomType, roomPrice, roomDescription)");
37              return ResponseEntity.status(response.getStatusCode()).body(response);
38          }
39
40          Response response = roomService.addNewRoom(photo, roomType, roomPrice, roomDescription);
41          return ResponseEntity.status(response.getStatusCode()).body(response);
42
43      }

```

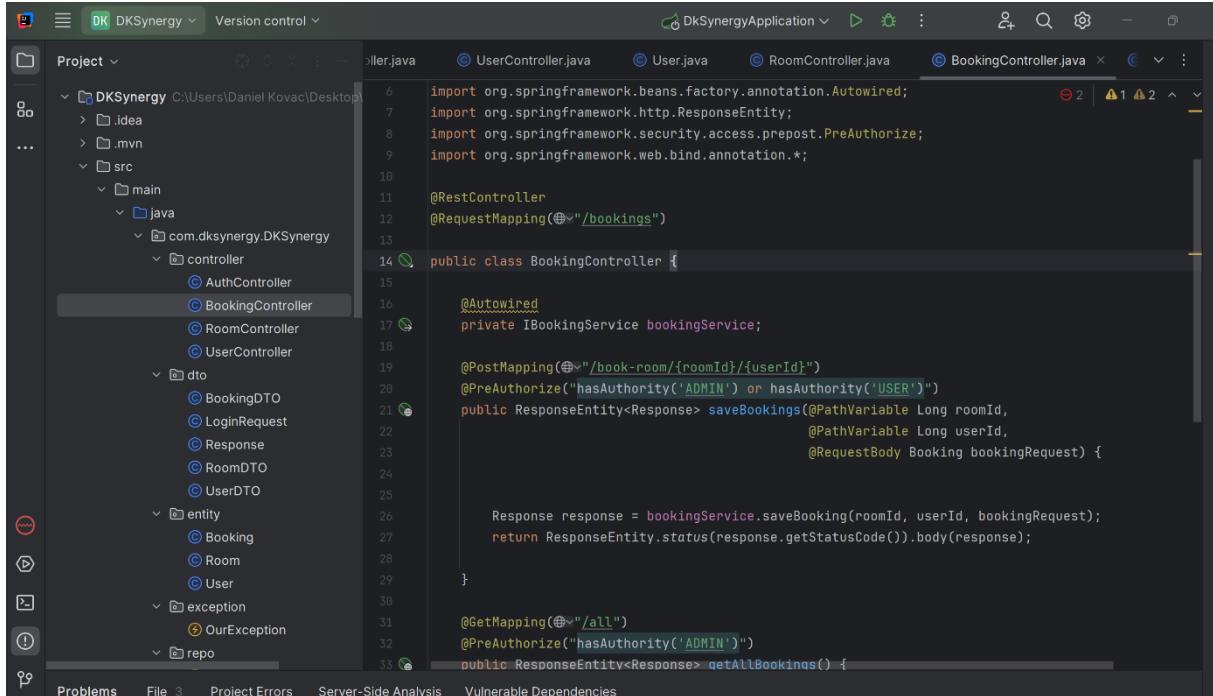
Zur Realisierung werden folgende Methoden genutzt:

1. **addNewRoom**
Fügt ein neues Zimmer hinzu.
2. **getAllRooms**
Gibt alle Zimmer zurück.
3. **getRoomTypes**
Gibt eine Liste aller verfügbaren Zimmertypen zurück.
4. **getRoomById**
Gibt die Details eines einzelnen Zimmers anhand seiner ID zurück.
5. **getAvailableRooms**
Gibt alle derzeit verfügbaren Zimmer zurück.
6. **getAvailableRoomsByDateAndType**
Gibt verfügbare Zimmer für einen bestimmten Zeitraum und Zimmertyp zurück.
7. **updateRoom**
Aktualisiert die Details eines Zimmers.
8. **deleteRoom**
Löscht ein Zimmer aus der Datenbank.

BookingController realisieren

Die **BookingController**-Klasse dient als zentrale Schnittstelle für Buchungsanfragen.

Sie verarbeitet HTTP-Anfragen für das Erstellen, Abrufen und Löschen von Buchungen und stellt sicher, dass nur autorisierte Benutzer darauf zugreifen können.



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under 'DKSynergy'. It includes a .idea folder, a .mvn folder, and a src folder containing main, com.dksynergy.DKSynergy, controller, dto, entity, exception, and repo packages.
- Code Editor:** Displays the `BookingController.java` file. The code implements a REST controller for bookings. It has two methods: `saveBookings` (POST /book-room/{roomId}/{userId}) and `getAllBookings` (GET /all). Both methods require 'ADMIN' or 'USER' authority.
- Toolbars and Status:** Standard IntelliJ toolbars and status bars are visible at the top and bottom of the interface.

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@.RestController
@RequestMapping("/bookings")
public class BookingController {

    @Autowired
    private IBookingService bookingService;

    @PostMapping("/book-room/{roomId}/{userId}")
    @PreAuthorize("hasAuthority('ADMIN') or hasAuthority('USER')")
    public ResponseEntity<Response> saveBookings(@PathVariable Long roomId,
                                                @PathVariable Long userId,
                                                @RequestBody Booking bookingRequest) {

        Response response = bookingService.saveBooking(roomId, userId, bookingRequest);
        return ResponseEntity.status(response.getStatusCode()).body(response);
    }

    @GetMapping("/all")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Response> getAllBookings() {
    }
}

```

Zur Realisierung werden folgende Methoden genutzt:

1. **saveBookings**

Erstellt eine neue Buchung für ein bestimmtes Zimmer und einen bestimmten Benutzer.

2. **getAllBookings**

Ruft alle Buchungen aus der Datenbank ab.

3. **getBookingByConfirmationCode**

Sucht eine Buchung anhand ihres BestätigungsCodes.

4. **cancelBooking**

Storniert eine Buchung basierend auf ihrer ID.

Backend testen

Testfall 1: User erstellen

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the package structure:
 - com.dksynergy
 - com.dksynergy.utils
 - com.dksynergy.aws
 - com.dksynergy.services
 - com.dksynergy.controllers
 - com.dksynergy.config
- Code Editor:** The current file is `DkSynergyApplication.java`, which contains the main method of the application.
- Run Configuration:** A run configuration for "DkSynergyApplication" is selected in the bottom left.
- Toolbars and Status Bar:** Standard IntelliJ IDEA toolbars and status bar showing the date and time.

Die User-Klasse definiert die erforderlichen Felder für die Benutzerregistrierung - E-Mail (unique), Name, Telefonnummer, Passwort und Rolle.

Wir wollen nun testen ob dies erfolgreich in der Datenbank gespeichert wird

Nachdem wir all dies erledigt haben, können wir mit den Tests beginnen.

```
    public class User implements UserDetails {
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @NotBlank(message = "E-Mail ist Notwendig")
        @Column(unique = true)
        private String email;

        @NotBlank(message = "Name ist Notwendig")
        private String name;

        @NotBlank(message = "Telefonnummer ist Notwendig")
        private String phoneNumber;

        @NotBlank(message = "Passwort ist Notwendig")
        private String password;

        private String role;

        @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
        private List<Booking> bookings = new ArrayList<>();
    }
}
```

Der Test der Benutzerregistrierung war erfolgreich: Der Admin-User "Max" wurde angelegt und das Passwort korrekt gehasht in der Datenbank gespeichert.

The screenshot shows two windows side-by-side. On the left is the Postman interface, displaying a successful POST request to 'localhost:4040/auth/register'. The request body contains a JSON object with fields: email ('Tony.Montana@gmail.com'), name ('Tony Montana'), phoneNumbers ('0794444444'), and password ('Tony'). The response is a 200 OK status with a response code of 208. On the right is the MySQL Workbench interface, showing the 'users' table in the 'modul223' schema. The table has columns: id, email, name, password, phone_number, and role. There are two rows: one for 'admin@gmail.com' (role: ADMIN) and one for 'test' (role: test). The bottom of the screen shows the Windows taskbar with various application icons.

Ein weiteres Beispiel zeigt die erfolgreiche Registrierung eines Benutzers mit der Rolle "USER", was die korrekte Funktionalität des Registrierungsprozesses bestätigt.

The screenshot shows the Postman interface. On the left, a POST request to `localhost:4040/signup` is being sent with a JSON body containing user registration data. On the right, a Result Grid displays the database table with two rows: one for the administrator and one for the registered user.

id	email	name	password	phone_number	role
1	admin@gmail.com	Max Mustermann	\$2a\$10\$Rx1hw3j18NU.Y9p80yahgetpJDLFrg...	0791234567	ADMIN
2	Tony.Montana@gmail.com	Tony Montana	\$2a\$10\$Wfw2Whx0410a4cfdzopeG.2xW6Eo...	0794444444	USER
	NULL	NULL	NULL	NULL	NULL

Testfall 2: User einloggen

The screenshot shows the Postman interface. A POST request to `localhost:4040/auth/login` is being sent with a JSON body containing the user's email and password. The response is a 200 OK status with a JSON object containing a JWT token, its expiration time, and the user's role.

```

{
  "statusCode": 200,
  "message": "erfolgreich",
  "token": "eyJhbGciOiJIUzI1NiJ9.\n27c20d013062551kvbRhnbfA2Z1hauwY29tLiiwiaHf0IjoxNzM0TA4NzAyLC1eHA10jE3Mzc5MTg3ODJ9.\nqfQXcvbd7jYOA-670udCCKVmnNF3ZGV-nV7m79g",
  "role": "USER",
  "expirationTime": "7 Tage"
}

```

Nach erfolgreicher Registrierung wurde der Login-Prozess getestet: Der Benutzer konnte sich mit seinen Anmeldedaten einloggen und erhielt ein gültiges JWT-Token mit 7 Tagen Gültigkeitsdauer zurück.

Testfall 3: Alle User abrufen

The screenshot shows the Postman interface. A GET request to `localhost:4040/users/all` is being sent with an Authorization header containing a Bearer Token. The response is a 200 OK status with a JSON object containing a list of users.

```

{
  "userList": [
    {
      "id": 1,
      "email": "admin@gmail.com",
      "name": "Max Mustermann",
      "phoneNumber": "0791234567",
      "role": "ADMIN",
      "bookings": []
    },
    {
      "id": 2,
      "email": "Tony.Montana@gmail.com",
      "name": "Tony Montana",
      "phoneNumber": "0794444444",
      "role": "USER",
      "bookings": []
    }
  ]
}

```

Below the screenshot is a snippet of Java code from the `UserService` class:

```

@GetMapping("/{all}")
@PreAuthorize("hasAuthority('ADMIN')")
public ResponseEntity<Response> getAllUsers() {
    Response response = userService.getAllUsers();
    return ResponseEntity.status(response.getStatusCode()).body(response);
}

```

Es wurde getestet, dass die Berechtigungsprüfung korrekt funktioniert: Der Endpunkt `/users/all` ist mit `@PreAuthorize("hasAuthority('ADMIN')")` geschützt und kann nur von Administratoren aufgerufen werden. Ein Benutzer mit der Rolle "USER" erhält wie vorgesehen eine Fehlermeldung.

Testfall 4: Bookings testen

```
@GetMapping("/get-user-bookings/{userId}")
public ResponseEntity<Response> getUserBookingHistory(@PathVariable("userId") String
    Response response = userService.getUserBookingHistory(userId);
    return ResponseEntity.status(response.getStatusCode()).body(response);
}
```

The screenshot shows a Postman request to `localhost:4040/users/get-user-bookings/1`. The response body is a JSON object:

```

1 {
  "statusCode": 200,
  "message": "erfolgreich",
  "user": {
    "id": 1,
    "email": "admin@gmail.com",
    "name": "Max Mustermann",
    "phoneNumber": "0791234567",
    "role": "ADMIN",
    "bookings": []
  }
}

```

Die Überprüfung des Buchungsendpunkts `/get-user-bookings/{userId}` zeigt die korrekte Funktionalität: Bei der Abfrage der Buchungshistorie eines Benutzers ohne Buchungen wird eine leere Buchungsliste zurückgegeben.

Testfall 5: User löschen

Fehlerbehandlung beim Löschen testen

The screenshot shows a Postman request to `localhost:4040/users/delete/3`. The response body is a JSON object:

```

1 {
  "statusCode": 404,
  "message": "Benutzer nicht gefunden"
}

```

Der erste Test zeigt die korrekte Fehlerbehandlung beim Löschen eines nicht existierenden Benutzers:

- Der API-Aufruf `/users/delete/3` gibt einen 404-Status zurück
- Die erwartete Fehlermeldung "Benutzer nicht gefunden" wird zurückgegeben

```

@Override 1 usage
public Response deleteUser(String userId) {

    Response response = new Response();

    try {
        userRepository.findById(Long.valueOf(userId)).orElseThrow(() -> new OurException("Benutzer nicht gefunden"));
        userRepository.deleteById(Long.valueOf(userId));
        response.setStatusCode(200);
        response.setMessage("erfolgreich");

    } catch (OurException e) {
        response.setStatusCode(404);
        response.setMessage(e.getMessage());
    }
}

```

Der Code im UserController zeigt die Implementierung:

- Suche des Benutzers über findById()
- Bei erfolgreichem Fund: Löschung und Statuscode 200
- Bei nicht gefundenem Benutzer: Exception mit Statuscode 404
- Try-Catch-Block für die Fehlerbehandlung

Richtiges löschen testen

The screenshot shows a Postman request to `localhost:4040/users/delete/2` using the `DELETE` method. The response status is `200 OK` and the message is `"erfolgreich"`.

Der zweite Test bestätigt die erfolgreiche Löschung eines existierenden Benutzers:

- Der API-Aufruf `/users/delete/2` gibt den Statuscode 200 zurück
- Die Erfolgsmeldung wird korrekt übermittelt

Die abschliessende SQL-Abfrage bestätigt:

- Der gelöschte Benutzer ist nicht mehr in der Datenbank vorhanden
- Andere Benutzer bleiben unverändert
- Die Datenbankintegrität bleibt erhalten

The screenshot shows the results of a `SELECT * FROM users` query in MySQL Workbench. The result grid contains one row of data:

	<code>id</code>	<code>email</code>	<code>name</code>	<code>password</code>	<code>phone_number</code>	<code>role</code>
1	HULL	admin@gmail.com	Max Mustermann	\$2a\$10\$Rx1hw3j18NU.Y9p80yahgetpJDLFFcg...	HULL	ADMIN

Testfall 6: Zimmer hinzufügen

Als Vorbereitung für die Tests haben wir mittels ChatGPT eine Liste verschiedener Hotelzimmerkategorien erstellen lassen. Für diese Kategorien wurden anschliessend über DALL-E passende Zimmerbilder generiert, die wir für die Visualisierung in unserem System verwenden konnten.

Es gibt eine ganze Reihe unterschiedlicher Hotelzimmerkategorien, die sich sowohl in ihrer Größe, Ausstattung als auch in ihrem Zweck unterscheiden. Im Folgenden ein Überblick über die gängigsten Zimmerarten:

- 1. Einzelzimmer (Single Room)
 - Für eine Person ausgelegt
 - Meist ein Einzelbett (manchmal auch ein kleines Doppelbett)
 - Günstige Option für Alleinreisende
- 2. Doppelzimmer (Double Room)
 - Für zwei Personen vorgesehen
 - Häufig mit einem Doppelbett oder Queensize-/Kingsize-Bett ausgestattet
 - Ideale Wahl für Paare oder Alleinreisende, die etwas mehr Platz möchten
- 3. Zweibettzimmer (Twin Room)
 - Zwei getrennte Einzelbetten statt einem Doppelbett
 - Praktisch für Freunde, Kollegen oder Gruppen, die sich ein Zimmer teilen möchten, aber getrennte Betten bevorzugen

Ask ChatGPT anything



Hier ist ein Bild von einem modernen Hotel-Einzelzimmer. Wenn Sie weitere Details oder Anpassungen wünschen, lassen Sie es mich wissen, und ich erstelle weitere für Sie!

Die API-Implementierung erfolgte über den RoomController, der das Hinzufügen neuer Zimmer ermöglicht. Dabei müssen verschiedene Parameter wie das Zimmerfoto, der Zimmertyp, der Preis und eine Beschreibung übermittelt werden. Durch die @PreAuthorize-Annotation wird sichergestellt, dass nur Administratoren neue Zimmer anlegen können.

```
27  
28     @PostMapping("/add")  
29     @PreAuthorize("hasAuthority('ADMIN')")  
30     public ResponseEntity<Response> addNewRoom(  
31         @RequestParam(value = "photo", required = false) MultipartFile photo,  
32         @RequestParam(value = "roomType", required = false) String roomType,  
33         @RequestParam(value = "roomPrice", required = false) BigDecimal roomPrice,  
34         @RequestParam(value = "roomDescription", required = false) String roomDescription  
35     ) {
```

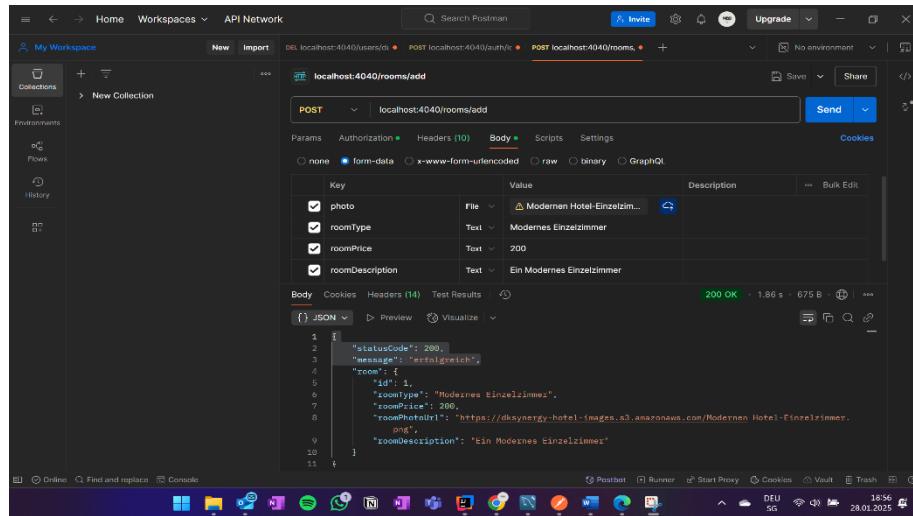
○ none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> photo	File <input type="button" value="△ Modernen Hotel-Einzelzim..."/>	<input type="button" value="Upload"/>		
<input checked="" type="checkbox"/> roomType	Text <input type="button" value="▼"/>	Modernes Einzelzimmer		
<input type="checkbox"/> roomPrice	Text <input type="button" value="▼"/>	200		
<input checked="" type="checkbox"/> roomDescription	Text <input type="button" value="▼"/>	Ein Modernes Einzelzimmer		

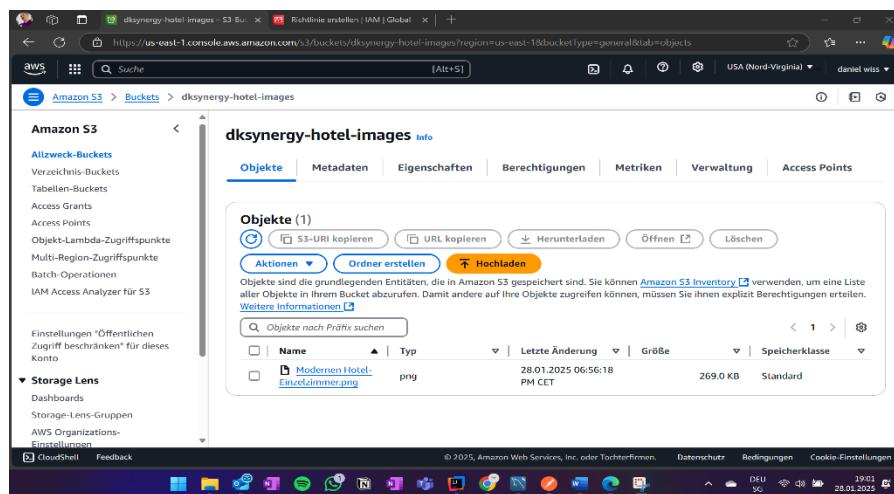
Body Cookies Headers (13) Test Results ⚡ 400 Bad Request • 16 ms • 502 B •

{ } JSON

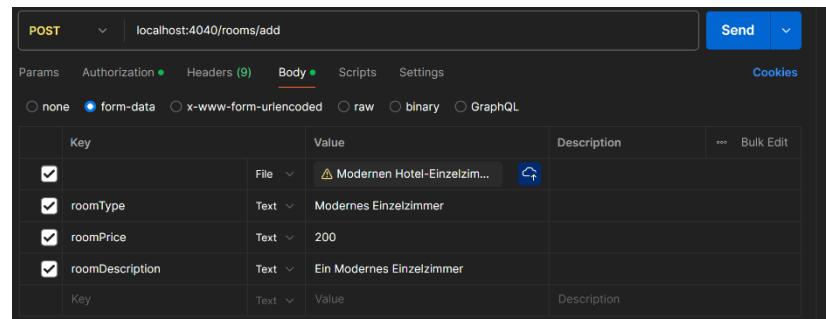
```
1 {  
2   "statusCode": 400,  
3   "message": "Bitte geben Sie Werte für alle Felder an(photo, roomType, roomPrice)"  
4 }
```



Der erste Test zur Zimmererstellung verlief erfolgreich. Das Zimmerfoto wurde korrekt im AWS S3 Bucket gespeichert und alle Zimmerinformationen wurden ordnungsgemäss in der Datenbank hinterlegt. Die API bestätigte den Erfolg mit einem 200 Statuscode und gab die gespeicherten Zimmerdetails zurück.



Um die Robustheit des Systems zu prüfen, wurde auch ein Test mit unvollständigen Daten durchgeführt. Wie erwartet reagierte das System mit einer entsprechenden Fehlerbehandlung.



Diese Tests bestätigen die korrekte Implementierung der Zimmer-Verwaltungsfunktionen, sowohl im Erfolgsfall als auch bei der Fehlerbehandlung.

Backend-Architektur

Das Backend des DKSynergy Hotels wurde mit Spring Boot entwickelt und repräsentiert eine robuste, skalierbare Architektur für das Hotelbuchungssystem. Der technologische Stack umfasst:

Technologische Grundlagen

- **Spring Boot 3.4.1:** Als primales Framework für die Anwendungsentwicklung
- **Java 23:** Programmiersprache mit modernen Sprachfeatures
- **Spring Security:** Für Authentifizierung und Autorisierung
- **JWT (JSON Web Token):** Für sichere Benutzeroauthentifizierung
- **MySQL:** Relationale Datenbank für Datenspeicherung
- **Hibernate/JPA:** Für Object-Relational Mapping
- **AWS S3:** Für Bildspeicherung und -verwaltung

Architekturelle Schichten

Die Backend-Architektur folgt einem klassischen Schichtenmodell, der die Verantwortlichkeiten klar definiert und voneinander trennt:

Controller-Schicht

Die Controller-Schicht bildet die Schnittstelle zwischen externen Anfragen und der Anwendungslogik:

- **UserController:** Verwaltet Benutzer-bezogene Endpunkte
- **RoomController:** Handling von Zimmer-Operationen
- **BookingController:** Steuerung von Buchungsprozessen
- **AuthenticationController:** Zuständig für Login und Registrierung

Service-Schicht

Implementiert die gesamte Geschäftslogik und fungiert als Vermittler zwischen Controllern und Repositories:

- **UserService:** Benutzer-Management und -Authentifizierung
- **RoomService:** Logik für Zimmer-Verwaltung
- **BookingService:** Buchungsprozesse und -validierung

Repository-Schicht

Verantwortlich für Datenbankzugriffe und Datenpersistenz:

- **UserRepository:** Datenbankoperationen für Benutzer
- **RoomRepository:** Zimmer-bezogene Datenbankabfragen
- **BookingRepository:** Buchungs-Datenbankinteraktionen

Sicherheitsarchitektur

Das Backend implementiert eine mehrschichtige Sicherheitsarchitektur:

JWT-basierte Authentifizierung

- Tokenbasierte Sicherheit mit JSON Web Tokens

- Stateless Authentication
- Rollenbasierte Zugriffskontrollen (ADMIN/USER)

Sicherheitskomponenten

- **SecurityConfig**: Zentrale Sicherheitskonfiguration
- **JWTAuthFilter**: Interceptor für Token-Validierung
- **JWTUtils**: Utility-Klasse für Token-Management
- **CustomUserDetailsService**: Benutzerspezifische Authentifizierungslogik

Externe Integrationen

AWS S3 Integration

- Speicherung von Zimmerbildern
- Generierung öffentlich zugänglicher URLs
- Effiziente Bildverwaltung ausserhalb der Hauptdatenbank

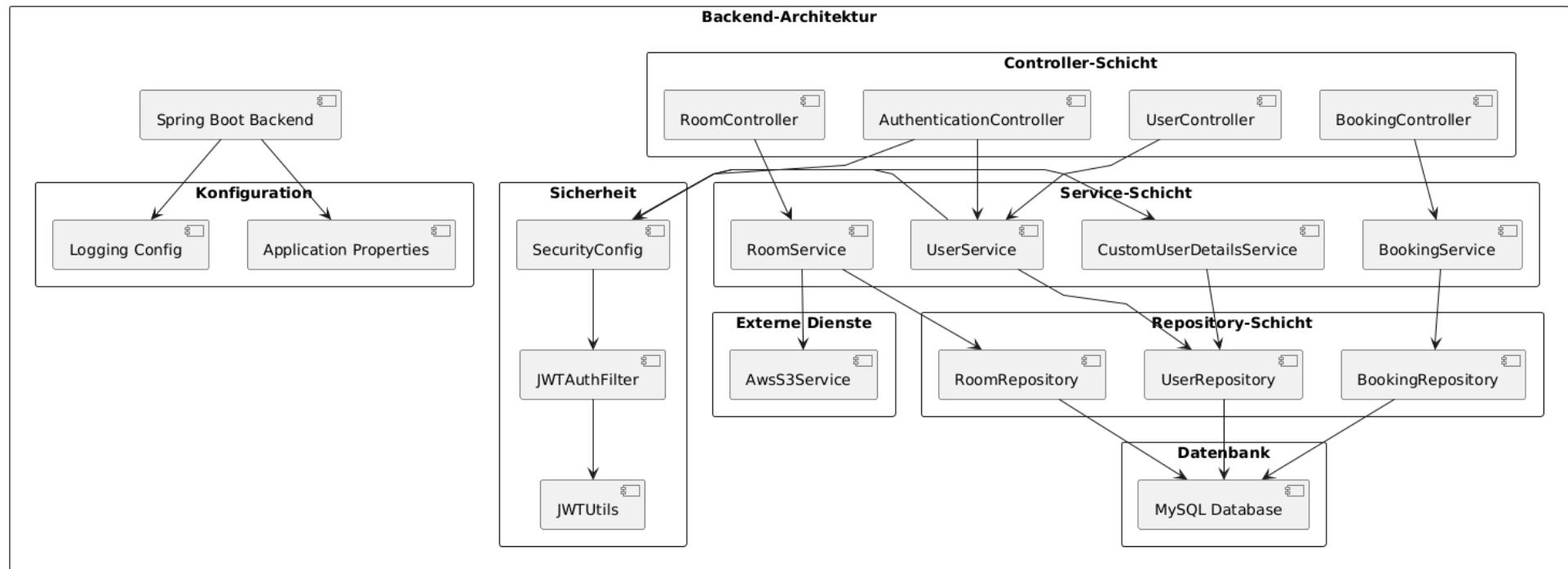
Datenmodell

Das Datenmodell basiert auf drei Hauptentitäten:

- **User**: Benutzerinformationen und Authentifizierungsdaten
- **Room**: Zimmerspezifische Informationen
- **Booking**: Buchungsdetails mit Referenzen zu Benutzern und Zimmern

Transaktionale Beziehungen

- Ein Benutzer kann mehrere Buchungen durchführen
- Ein Zimmer kann mehrere zeitlich getrennte Buchungen haben
- Jede Buchung speichert wesentliche Informationen wie Check-in/out-Daten



Realisation des Front-Ends

Initialisierung des Projekts

```
PS C:\Users\Daniel Kovac\Desktop> npm install react react-dom react-scripts cra-template
Microsoft Windows [Version 10.0.22631.4751]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Windows\System32>cd C:\Users\Daniel Kovac\Desktop

C:\Users\Daniel Kovac\Desktop>npx create-react-app dksynergy-hotel-react

Creating a new React app in C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

|.
```

Bei der initialen Entwicklung des Frontends für das Hotelbuchungssystem traten wir zunächst auf technische Herausforderungen bei der Erstellung einer React-Anwendung. Die Entwicklungsumgebung generierte konsistent den folgenden Fehler:

```
npm ERR! ERESOLVE unable to resolve dependency tree
```

Nach ausführlicher Recherche auf Entwicklerplattformen wie Stack Overflow und Reddit wurde ersichtlich, dass dies ein bekanntes Problem seit React 19 ist. Ein Downgrade auf React 18 erwies sich als nicht praktikabel, da das System automatisch wieder auf Version 19 aktualisierte.

Als Lösung entschieden wir uns für die Migration zu Vite als Build-Tool in Kombination mit React. Die Installation erfolgte erfolgreich mit dem Befehl:

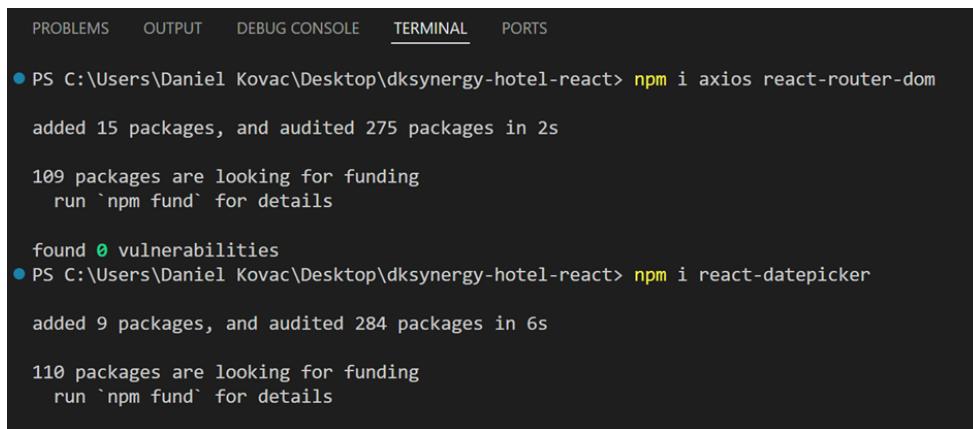
```
PS C:\Users\Daniel Kovac> cd Desktop
PS C:\Users\Daniel Kovac\Desktop> npx create vite@latest dksynergy-hotel-react -- --template react
Need to install the following packages:
create-vite@6.1.1
Ok to proceed? (y) y

> npx
> create-vite dksynergy-hotel-react --template react

Scaffolding project in C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react...
Done. Now run:

  cd dksynergy-hotel-react
  npm install
  npm run dev

PS C:\Users\Daniel Kovac\Desktop> |
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> npm i axios react-router-dom
added 15 packages, and audited 275 packages in 2s

109 packages are looking for funding
  run `npm fund` for details

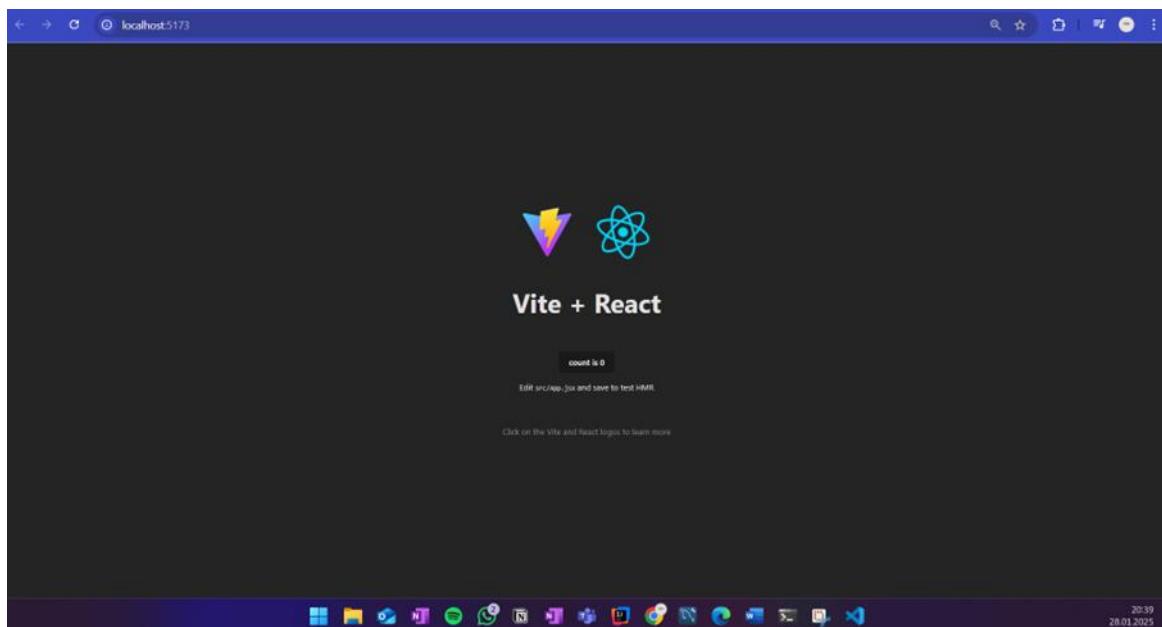
found 0 vulnerabilities
● PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> npm i react-datepicker
added 9 packages, and audited 284 packages in 6s

110 packages are looking for funding
  run `npm fund` for details
```

Anschliessend wurden folgende essenzielle Pakete für die Entwicklung installiert:

- axios für HTTP-Anfragen
- react-router-dom für das Routing
- react-datepicker für Datumsauswahl-Funktionalitäten

Die Entwicklungsumgebung konnte nach der Installation erfolgreich gestartet werden und läuft auf localhost:5173.



Abhängigkeiten installieren

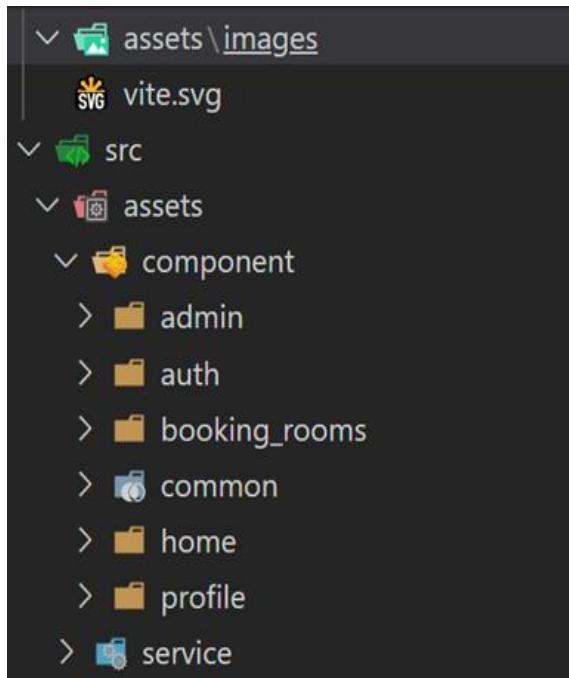
Nach der erfolgreichen Erstellung des React-Projekts wurden zusätzliche Pakete installiert, die für die Funktionalität der Anwendung erforderlich sind.

1. Installation von axios und react-router-dom
 - Axios wird für HTTP-Anfragen genutzt, um mit der Spring Boot API zu kommunizieren.
 - React Router DOM wird für die Navigation innerhalb der Anwendung benötigt.

2. Installation von react-datepicker
 - Dieses Paket wird für die Datumsauswahl verwendet.

Erstellung der Verzeichnisstruktur im Frontend

Um eine saubere und skalierbare Architektur für das React-Frontend zu gewährleisten, wurden verschiedene Ordner erstellt.



Kernverzeichnisse und deren Funktionen

1. components/

Enthält modulare, wiederverwendbare UI-Komponenten:

- admin/: Administrative Funktionalitäten und Verwaltungsoberflächen
- auth/: Authentifizierungskomponenten
- booking_rooms/: Komponenten für die Zimmerbuchung
- common/: Gemeinsam genutzte Basiskomponenten
- home/: Komponenten der Startseite
- profile/: Benutzerprofil-bezogene Komponenten

2. assets/

Verwaltet statische Ressourcen:

- images/: Bildmaterial und Icons
- Speicherort für Design-Assets und visuelle Elemente

3. services/

Implementiert die Backend-Kommunikation.

Zusätzliche Verzeichnisse

- context/: Global verfügbare Zustandsverwaltung
- utils/: Hilfsfunktionen und gemeinsam genutzte Logik

Zugriffskontrolle und API-Kommunikation im Frontend

Um sicherzustellen, dass nur berechtigte Benutzer bestimmte Bereiche der Anwendung nutzen können und um eine saubere Kommunikation mit dem Backend zu ermöglichen, wurden zwei zentrale Module implementiert:

1. Zugriffskontrolle mit guard.js

- Schutz vor unbefugtem Zugriff auf geschützte Seiten (z. B. Admin-Dashboard).
 - ProtectedRoute prüft, ob ein Nutzer eingeloggt ist. Falls nicht, wird zur Login-Seite umgeleitet.
- Trennung zwischen normalen Nutzern und Administratoren.
 - AdminRoute stellt sicher, dass nur Administratoren auf bestimmte Seiten zugreifen können.

2. API-Kommunikation mit ApiService.js

- Einheitliche und zentrale Verwaltung aller API-Aufrufe.
 - Methoden zur Authentifizierung (registerUser, loginUser).
- Automatische Einbindung des Tokens für authentifizierte Anfragen.
 - Abruf von Benutzer-, Zimmer- und Buchungsdaten (getAllRooms, bookRoom, getUserProfile).
 - Zugriffsprüfung (isAuthenticated, isAdmin).

```

src > assets > component > common > Navbar.jsx
  1 import React from 'react';
  2 import { NavLink, useNavigate } from 'react-router-dom';
  3 import ApiService from '../../../../../service/ApiService';
  4
  5 function Navbar() {
  6   const isAuthenticated = ApiService.isAuthenticated();
  7   const isAdmin = ApiService.isAdmin();
  8   const isUser = ApiService.isUser();
  9   const navigate = useNavigate();
 10
 11   // Event-Handler für den Logout-Prozess
 12   const handleLogout = () => {
 13     // Bestätigungs Nachricht auf Deutsch
 14     const isLogout = window.confirm("Sind Sie sicher, dass Sie diesen Benutzer abmelden möchten?");
 15     if (isLogout) {
 16       ApiService.logout();
 17       navigate('/home');
 18     }
 19   };
 20
 21   return (
 22     <nav className="navbar">
 23       <div className="navbar-brand">
 24         <NavLink to="/home" DKSynergy Hotel</NavLink>
 25       </div>
 26       <ul className="navbar-ul">
 27         <li><NavLink to="/home" activeclassname="active">Startseite</NavLink></li>
 28         <li><NavLink to="/rooms" activeclassname="active">Zimmer</NavLink></li>
 29       </ul>
 30     </nav>
 31   );
 32 }
 33
 34 export default Navbar;
 35
 36 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> []

Ln 40, Col 2 Spaces: 4 UTF-8 CRLF JavaScript JSX Go Live Go Live

DEU SG 23:21 30.01.2025

Web-Pages realisieren

Navigationsleiste (Navbar.js) erstellen

Die Navigation wurde als responsive Komponente implementiert, die sich dynamisch an den Benutzerkontext anpasst:

Rollenbasierte Darstellung:

- Nicht authentifizierte Benutzer: Startseite, Zimmerübersicht, Login/Registrierung
 - Authentifizierte Benutzer: Zusätzlich Profilzugriff und Logout-Option
 - Administratoren: Erweiterte Verwaltungsfunktionen

Sicherheitsfunktionen:

- Integrierte Logout-Funktionalität mit Bestätigungsdialog
 - Automatische Token-Entfernung und Redirect zur Startseite
 - Zugriffssteuerung für geschützte Routen

The screenshot shows the VS Code interface with the following details:

- File Explorer:** On the left, it shows the project structure under "OPEN EDITORS". The "src" folder contains "assets", "component", "home", and "service" subfolders. "home" contains "Footer.jsx", "Navbar.jsx", and "HomePage.jsx".
- Code Editor:** The main area displays the "HomePage.jsx" file. The code is a React component named "HomePage". It includes sections for the header/banner, a banner section with a banner image and an overlay, and a service section with a service card.
- Bottom Bar:** The bottom bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. It also features a status bar with "Ln 46, Col 23", "Spaces: 4", "UTF-8", "CRLF", "JavaScript JSX", and "Go Live" buttons.

Ein standardisierter Footer wurde ebenfalls implementiert, der auf allen Seiten konsistent dargestellt wird.

Startseite (HomePage.js) erstellen

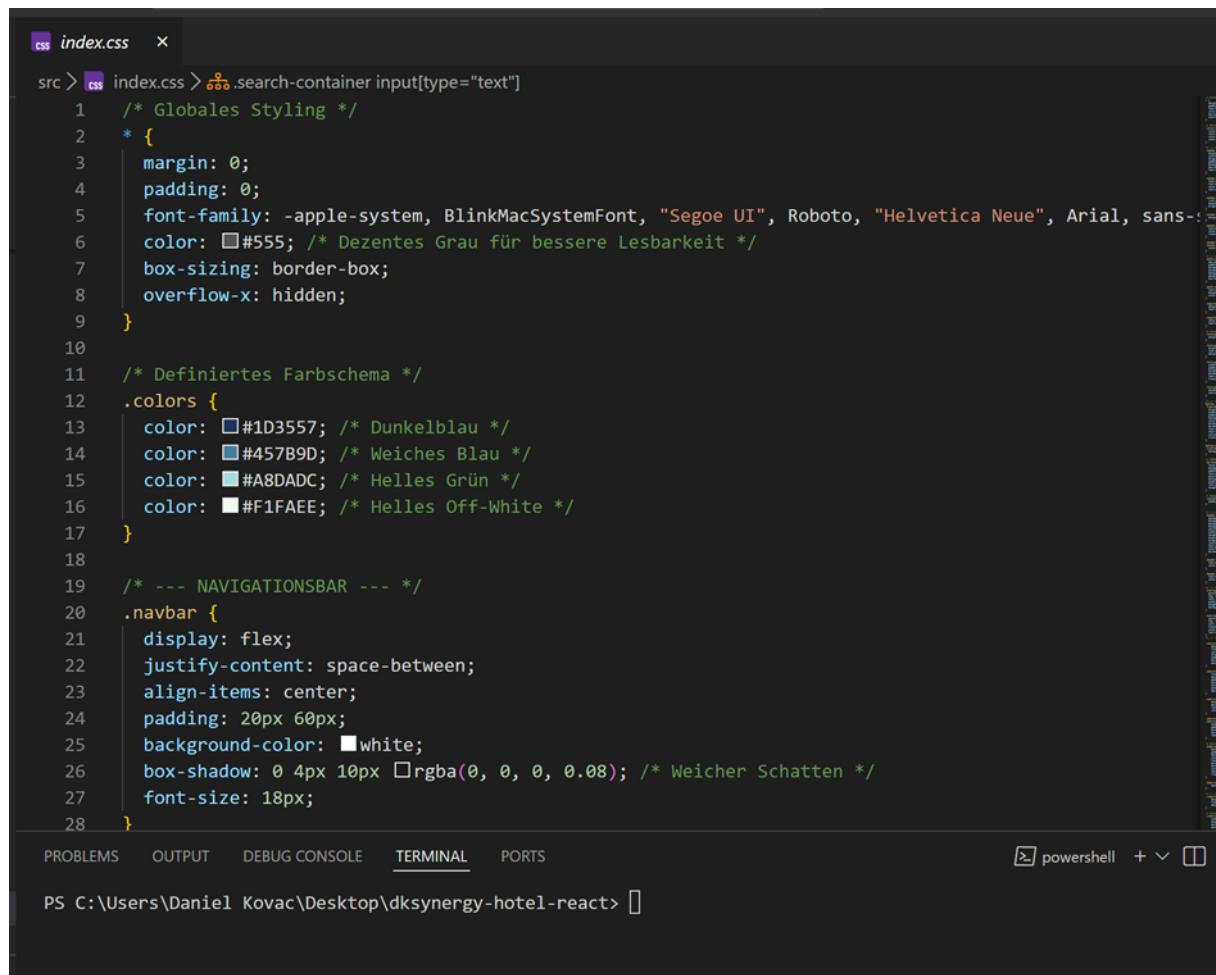
Die Startseite ist der erste Berührungsplatz für Besucher der Anwendung. Sie bietet eine Übersicht über das Hotel, seine Dienstleistungen und ermöglicht eine direkte Suche nach verfügbaren Zimmern.

Hier sind einige Funktionalitäten der Startseiter:

1. Header mit Begrüßungstext
2. Zimmer-Suchfunktion (RoomSearch)
3. Link zu allen Zimmern
4. Dienstleistungen des Hotels

Für das Frontend-Design wurde ein modernes und responsives Layout erstellt. Dabei wurde das CSS aus einem früheren Projekt (Modul 295) wiederverwendet, angepasst und mit zusätzlichen Stilelementen erweitert.

Zusätzlich wurden CSS-Codes aus Online-Quellen recherchiert und optimiert. ChatGPT wurde genutzt, um bestehende CSS-Strukturen zu verbessern und einheitliches Design zu gewährleisten.



```

css index.css x
src > index.css > search-container input[type="text"]
1  /* Globales Styling */
2  * {
3    margin: 0;
4    padding: 0;
5    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Arial, sans-serif;
6    color: #555; /* Dezentes Grau für bessere Lesbarkeit */
7    box-sizing: border-box;
8    overflow-x: hidden;
9  }
10 /* Definiertes Farbschema */
11 .colors {
12   color: #1D3557; /* Dunkelblau */
13   color: #457B9D; /* Weiches Blau */
14   color: #A8DADC; /* Helles Grün */
15   color: #F1FAEE; /* Helles Off-White */
16 }
17
18 /* --- NAVIGATIONSBAR --- */
19 .navbar {
20   display: flex;
21   justify-content: space-between;
22   align-items: center;
23   padding: 20px 60px;
24   background-color: white;
25   box-shadow: 0 4px 10px rgba(0, 0, 0, 0.08); /* Weicher Schatten */
26   font-size: 18px;
27 }
28

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react>

Routing- und Seitenstruktur (App.jsx) erstellen

Die App.jsx fungiert als zentrale Routing-Komponente und implementiert die grundlegende Navigationslogik der Anwendung.

```

App.jsx
src > App.jsx > fApp
14 import ManageRoomPage from './assets/component/admin/ManageRoomPage.jsx';
15 import EditRoomPage from './assets/component/admin/EditRoomPage.jsx';
16 import AddRoomPage from './assets/component/admin/AddRoomPage.jsx';
17 import ManageBookingsPage from './assets/component/admin/ManageBookingsPage.jsx';
18 import EditBookingPage from './assets/component/admin/EditBookingPage.jsx';
19 import ProfilePage from './assets/component/profile/ProfilePage.jsx';
20 import EditProfilePage from './assets/component/profile/EditProfilePage.jsx';
21 import { ProtectedRoute, AdminRoute } from './assets/service/guard.jsx'; // Stellen Sie sicher, dass Sie den Pfad ändern
22
23 function App() {
24   return (
25     <BrowserRouter>
26       <div className="App">
27         <Navbar />
28         <div className="content">
29           <Routes>
30             {/* Öffentliche Routen */}
31             <Route exact path="/home" element={<HomePage />} />
32             <Route exact path="/login" element={<LoginPage />} />
33             <Route path="/register" element={<RegisterPage />} />
34             <Route path="/rooms" element={<AllRoomsPage />} />
35             <Route path="/find-booking" element={<FindBookingPage />} />
36
37             {/* Geschützte Routen */}
38             <Route path="/room-details-book/:roomId"
39               element={<ProtectedRoute element={<RoomDetailsBookingPage />} />}
40             />
41             <Route path="/profile"
42           />
43           <Route path="/" element={<HomePlaceholder />} />
44         </Routes>
45       </div>
46     </div>
47   )
48 }

```

Sicherheitskomponenten

1. Zugriffssteuerung
 - o ProtectedRoute: Authentifizierungscheck für geschützte Bereiche
 - o AdminRoute: Zusätzliche Autorisierungsprüfung für administrative Funktionen

Routing-Hierarchie

1. Öffentliche Routen
 - o Homepage
 - o Login/Registrierung
 - o Zimmerübersicht
2. Authentifizierte Routen
 - o Profilbereich
 - o Buchungsverwaltung
 - o Zimmerdetails
3. Administrative Routen
 - o Zimmerverwaltung
 - o Buchungsmanagement
 - o Benutzerverwaltung

Fehlerbehandlung

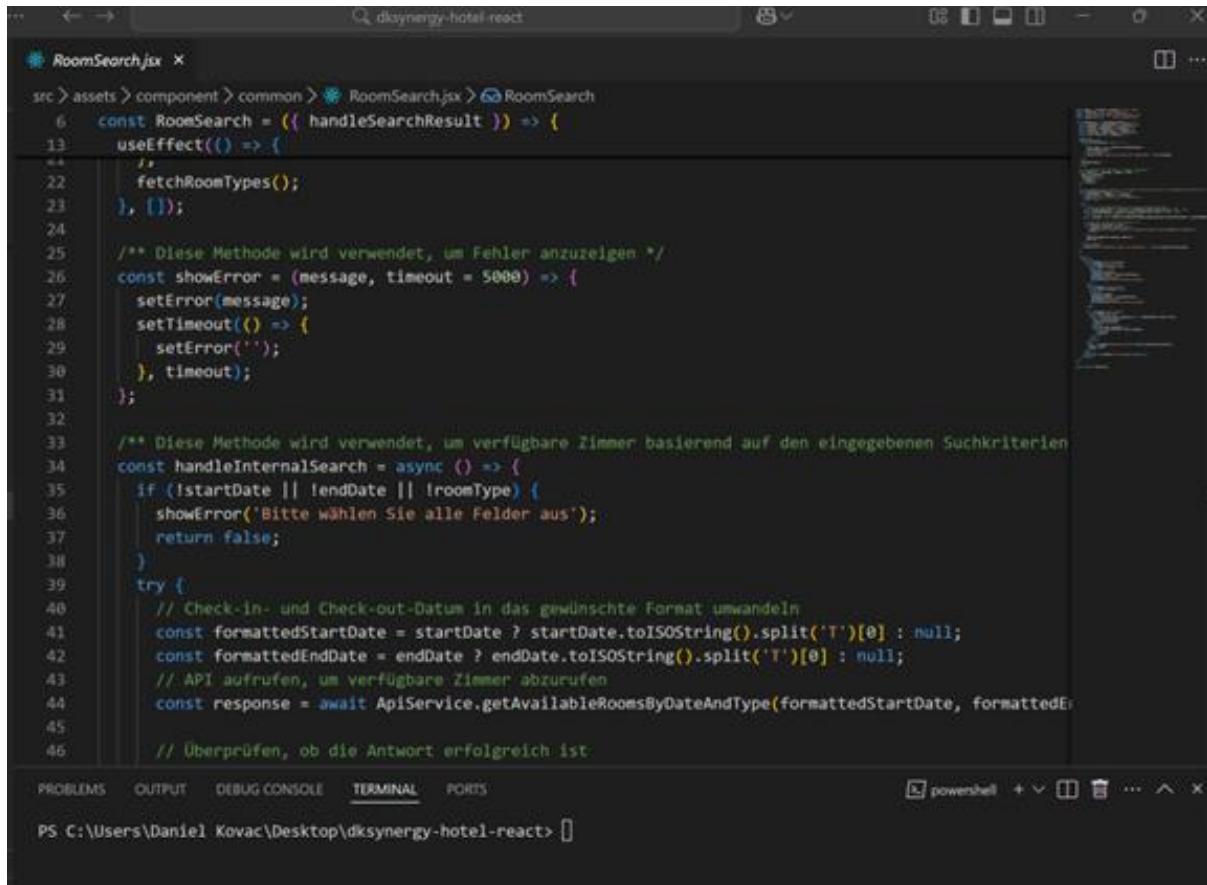
- o Implementierung eines 404-Fallbacks
- o Automatische Weiterleitung bei ungültigen URLs
- o Sicherheitsrerouting bei fehlenden Berechtigungen

Diese Architektur gewährleistet eine robuste Navigation und effektive Zugriffskontrolle innerhalb der Anwendung.

immer-Suchfunktion (RoomSearch.js) erstellen

Diese Komponente ermöglicht es den Nutzern, verfügbare Zimmer basierend auf Check-in- und Check-out-Datum sowie Zimmertyp zu suchen.

Sie bietet eine dynamische, interaktive Möglichkeit, Zimmer aus der Datenbank abzurufen und stellt sicher, dass nur gültige Suchanfragen gesendet werden.



The screenshot shows a code editor window with the file 'RoomSearch.jsx' open. The code is written in JavaScript and uses the React framework. It includes logic for fetching room types, handling search errors, and performing an internal search based on start date, end date, and room type. The code editor has a dark theme and displays line numbers from 1 to 46. Below the code editor is a terminal window showing the command 'PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> []'. The terminal also shows the path 'src > assets > component > common > RoomSearch.jsx'.

```
... ← → 🔍 dksynergy-hotel-react
RoomSearch.jsx ×
src > assets > component > common > RoomSearch.jsx
6  const RoomSearch = ({ handleSearchResult }) => {
13    useEffect(() => {
14      // ...
15      fetchRoomTypes();
16    }, [ ]);
17
18    /** Diese Methode wird verwendet, um Fehler anzuzeigen */
19    const showError = (message, timeout = 5000) => {
20      setError(message);
21      setTimeout(() => {
22        setError('');
23      }, timeout);
24    };
25
26    /** Diese Methode wird verwendet, um verfügbare Zimmer basierend auf den eingegebenen Suchkriterien
27    zu suchen */
28    const handleInternalSearch = async () => {
29      if (!startDate || !endDate || !roomType) {
30        showError('Bitte wählen Sie alle Felder aus');
31        return false;
32      }
33      try {
34        // Check-in- und Check-out-Datum in das gewünschte Format umwandeln
35        const formattedStartDate = startDate ? startDate.toISOString().split('T')[0] : null;
36        const formattedEndDate = endDate ? endDate.toISOString().split('T')[0] : null;
37        // API aufrufen, um verfügbare Zimmer abzurufen
38        const response = await ApiService.getAvailableRoomsByDateAndType(formattedStartDate, formattedEndD...
39
40        // Überprüfen, ob die Antwort erfolgreich ist
41        if (response.ok) {
42          handleSearchResult(response.data);
43        } else {
44          showError(`Fehler: ${response.statusText}`);
45        }
46      } catch (error) {
47        showError(`Fehler: ${error.message}`);
48      }
49    };
50
51    handleInternalSearch();
52  }
53
54  return (
55    <div>
56      <h2>Suchen</h2>
57      <Formik
58        initialValues={initialValues}
59        validationSchema={validationSchema}
60        onSubmit={handleSearch}
61      >
62        <Form>
63          <Grid>
64            <GridItem>
65              <DatePicker
66                value={startDate}
67                onChange={setStartDate}
68                label="Check-in"
69              />
70            </GridItem>
71            <GridItem>
72              <DatePicker
73                value={endDate}
74                onChange={setEndDate}
75                label="Check-out"
76              />
77            </GridItem>
78            <GridItem>
79              <Select
80                value={roomType}
81                onChange={setRoomType}
82                options={roomTypes}
83                label="Zimmertyp"
84              />
85            </GridItem>
86            <GridItem>
87              <button type="submit">Suchen</button>
88            </GridItem>
89          </Grid>
90        </Form>
91      </Formik>
92    </div>
93  );
94
95  export default RoomSearch;
```

Suchergebnisseite (RoomResult.jsx) erstellen

Die RoomResult-Komponente fungiert als zentrale Schnittstelle zur Darstellung von Suchergebnissen im Hotelbuchungssystem. Als direkte Erweiterung der RoomSearch-Komponente verarbeitet sie die über handleSearchResult übermittelten Suchergebnisse.

Die Komponente rendert eine Kartenansicht verfügbarer Zimmer mit relevanten Informationen wie Bildern, Kategorien, Preisen und Beschreibungen. Durch rollenbasierte Navigation erhalten reguläre Nutzer Zugang zur Buchungsfunktion, während Administratoren zusätzliche Bearbeitungsoptionen sehen.

Die Integration des `useNavigate`-Hooks ermöglicht eine nahtlose Weiterleitung zu Detail- und Buchungsseiten, wodurch ein effizienter und benutzerfreundlicher Buchungsprozess gewährleistet wird.

RoomResult.jsx

```
src > assets > component > common > RoomResult.jsx > RoomResult > roomSearchResults.map() callback
1 import React from 'react';
2 import { useNavigate } from 'react-router-dom'; // Importiere useNavigate
3 import ApiService from '../../../../../service/ApiService';
4
5 const RoomResult = ({ roomSearchResults }) => {
6     const navigate = useNavigate(); // Initialisiere useNavigate-Hook
7     const isAdmin = ApiService.isAdmin();
8
9     return (
10         <section className="room-results">
11             {roomSearchResults && roomSearchResults.length > 0 && (
12                 <div className="room-list">
13                     {roomSearchResults.map(room => (
14                         <div key={room.id} className="room-list-item">
15                             <img className="room-list-item-image" src={room.roomPhotoUrl} alt={room.roomType}>
16                             <div className="room-details">
17                                 <h3>{room.roomType}</h3>
18                                 <p>Preis: €{room.roomPrice} / Nacht</p>
19                                 <p>Beschreibung: {room.roomDescription}</p>
20                             </div>
21
22                             <div className='book-now-div'>
23                                 {isAdmin ? (
24                                     <button
25                                         className="edit-room-button"
26                                         onClick={() => navigate(`admin/edit-room/${room.id}`)} // Navigation
27                                     >
28                                         Raum bearbeiten
29                                 ) : (
30                                     <button
31                                         className="book-now-button"
32                                         onClick={() => handleBookNow(room.id)} // Book now logic
33                                     >
34                                         Jetzt buchen
35                                 )
36                             </div>
37                         </div>
38                     ))}
39                 </div>
40             )}
41         </section>
42     );
43 }
44
45 export default RoomResult;
```

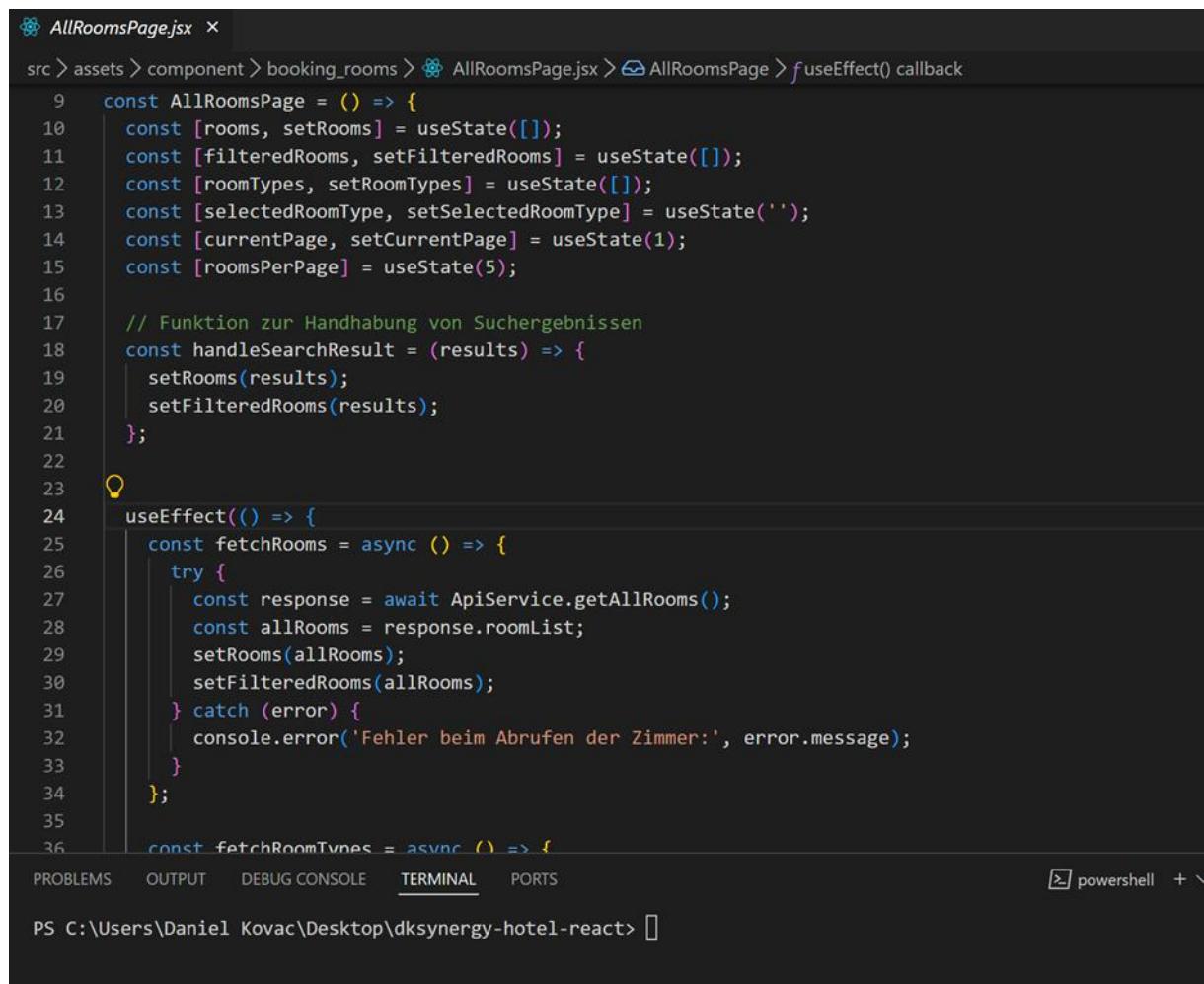
Zimmeransichtseite (AllRoomsPage.jsx) erstellen

Die AllRoomsPage fungiert als zentrale Komponente zur Verwaltung und Anzeige der Hotelzimmer. Sie integriert mehrere React-Hooks für das State-Management.

Die Komponente implementiert eine asynchrone Datenabruflogik mittels useEffect für Zimmer und Zimmertypen, sowie Funktionen zur Filterung und Ergebnisverarbeitung.

Die Pagination-Komponente ermöglicht eine effiziente Navigation durch die Zimmerliste mit folgenden Kernfunktionen:

- Berechnung der Gesamtseitenzahl basierend auf Zimmermenge und Anzeigelimit
- Dynamische Seitennavigation
- Visuelle Hervorhebung der aktiven Seite
- Optimierte Ladeleistung durch begrenzte Datenmenge pro Seite



```

    const AllRoomsPage = () => {
      const [rooms, setRooms] = useState([]);
      const [filteredRooms, setFilteredRooms] = useState([]);
      const [roomTypes, setRoomTypes] = useState([]);
      const [selectedRoomType, setSelectedRoomType] = useState('');
      const [currentPage, setCurrentPage] = useState(1);
      const [roomsPerPage] = useState(5);

      // Funktion zur Handhabung von Suchergebnissen
      const handleSearchResult = (results) => {
        setRooms(results);
        setFilteredRooms(results);
      };

      useEffect(() => {
        const fetchRooms = async () => {
          try {
            const response = await ApiService.getAllRooms();
            const allRooms = response.roomList;
            setRooms(allRooms);
            setFilteredRooms(allRooms);
          } catch (error) {
            console.error('Fehler beim Abrufen der Zimmer:', error.message);
          }
        };
        const fetchRoomTypes = async () => {
          ...
        };
      }, []);
    }
  
```

Die Integration dieser Komponenten resultiert in einer benutzerfreundlichen Oberfläche, die effiziente Zimmersuche und -filterung ermöglicht, während die Systemleistung durch optimierte Datenladestrategien gewährleistet bleibt.

Buchungssuchseite (FindBookingPage.jsx) erstellen

Die FindBookingPage-Komponente wurde entwickelt, um Gästen einen schnellen, unkomplizierten Zugriff auf ihre Buchungsinformationen zu ermöglichen. Die Komponente basiert auf React-Hooks für effizientes State-Management.

Kernfunktionalitäten sind:

1. Buchungsabruf

Asynchrone API-Kommunikation via ApiService.getBookingByConfirmationCode()

2. Validierung

Prüfung auf gültige Eingabe des BestätigungsCodes

3. Fehlerbehandlung

Temporäre Fehleranzeige mit 5-Sekunden-Timeout

4. Detailansicht

Strukturierte Darstellung der Buchungsinformationen

```
const FindBookingPage = () => {
  const [confirmationCode, setConfirmationCode] = useState(''); // Zustandsvariable für Bestätigungscode
  const [bookingDetails, setBookingDetails] = useState(null); // Zustandsvariable für Buchungsdetails
  const [error, setError] = useState(null); // Fehler verfolgen

  const handleSearch = async () => {
    if (!confirmationCode.trim()) {
      setError("Bitte geben Sie einen Buchungsbestätigungscode ein");
      setTimeout(() => setError(null), 5000);
      return;
    }
    try {
      // API aufrufen, um Buchungsdetails zu erhalten
      const response = await ApiService.getBookingByConfirmationCode(confirmationCode);
      setBookingDetails(response.booking);
      setError(null); // Fehler bei Erfolg löschen
    } catch (error) {
      setError(error.response?.data?.message || error.message);
      setTimeout(() => setError(null), 5000);
    }
  };

  return (
    <div className="find-booking-page">
      <h2>Buchung finden</h2>

```

Zimmerdetailsseite (RoomDetailsPage.jsx) erstellen

Die RoomDetailsPage fungiert als zentrale Buchungsschnittstelle des Hotelsystems. Sie vereint Zimmerdetails und Buchungsprozess in einer optimierten Benutzeroberfläche.

Bei Aufruf lädt die Komponente automatisch Zimmerinformationen, prüft Verfügbarkeiten und ermöglicht die direkte Buchungsabwicklung. Nutzer können Aufenthaltszeitraum und Gästeanzahl festlegen, wobei der Gesamtpreis dynamisch berechnet wird. Die integrierte Validierung prüft alle Eingaben und gibt unmittelbares Feedback. Nach erfolgreicher Buchung erfolgt die automatische Bestätigungsgenerierung mit Buchungscode.

Diese effiziente Integration aller buchungsrelevanten Funktionen in einer Komponente optimiert den Buchungsprozess und steigert die Benutzerfreundlichkeit erheblich.

```
const FindBookingPage = () => {
  const [confirmationCode, setConfirmationCode] = useState(''); // Zustandsvariable für Bestätigungscode
  const [bookingDetails, setBookingDetails] = useState(null); // Zustandsvariable für Buchungsdetails
  const [error, setError] = useState(null); // Fehler verfolgen

  const handleSearch = async () => {
    if (!confirmationCode.trim()) {
      setError("Bitte geben Sie einen Buchungsbestätigungscode ein");
      setTimeout(() => setError(null), 5000);
      return;
    }
    try {
      // API aufrufen, um Buchungsdetails zu erhalten
      const response = await ApiService.getBookingByConfirmationCode(confirmationCode);
      setBookingDetails(response.booking);
      setError(null); // Fehler bei Erfolg löschen
    } catch (error) {
      setError(error.response?.data?.message || error.message);
      setTimeout(() => setError(null), 5000);
    }
  };

  return (
    <div className="find-booking-page">
      <h2>Buchung finden</h2>
      ...
    </div>
  );
}
```

Anmeldeseite (LoginPage.jsx) erstellen

Die LoginPage.jsx-Komponente ermöglicht es Benutzern, sich in das Hotelbuchungssystem einzuloggen.

Das Frontend-Authentifizierungssystem verwaltet Benutzerzugriffe durch Token- und Rollenspeicherung im localStorage. Es ermöglicht persistente Anmeldungen und steuert den Zugang zu geschützten Bereichen. Nach der Authentifizierung erfolgt eine automatische Weiterleitung zur ursprünglich angefragten Seite, was eine nahtlose Benutzerführung gewährleistet.

Es ermöglicht folgendes:

1. Benutzer gibt E-Mail und Passwort ein
 - Eingaben werden im State (useState) gespeichert.
 - Falls ein Feld leer bleibt, wird eine Fehlermeldung ausgegeben.
2. Anmeldeprozess über ApiService.loginUser()
 - Falls erfolgreich:
 - Der Token und die Benutzerrolle werden in localStorage gespeichert.
 - Der Benutzer wird zur letzten Seite weitergeleitet oder zur Startseite (/home).
 - Falls fehlerhaft:
 - Eine Fehlermeldung wird angezeigt und nach 5 Sekunden automatisch gelöscht.
3. Visuelles Feedback für den Nutzer
 - Fehlermeldung bei falschen Eingaben
 - Link zur Registrierung für neue Benutzer

```

function LoginPage() {
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();
  const location = useLocation();

  const from = location.state?.from?.pathname || '/home';

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!email || !password) {
      setError('Bitte füllen Sie alle Felder aus.');
      setTimeout(() => setError(''), 5000);
      return;
    }

    try {
      const response = await ApiService.loginUser({email, password});
      if (response.statusCode === 200) {
        localStorage.setItem('token', response.token);
        localStorage.setItem('role', response.role);
        navigate(from, { replace: true });
      }
    } catch (error) {
      setError(error.response?.data?.message || error.message);
      setTimeout(() => setError(''), 5000);
    }
  }
}

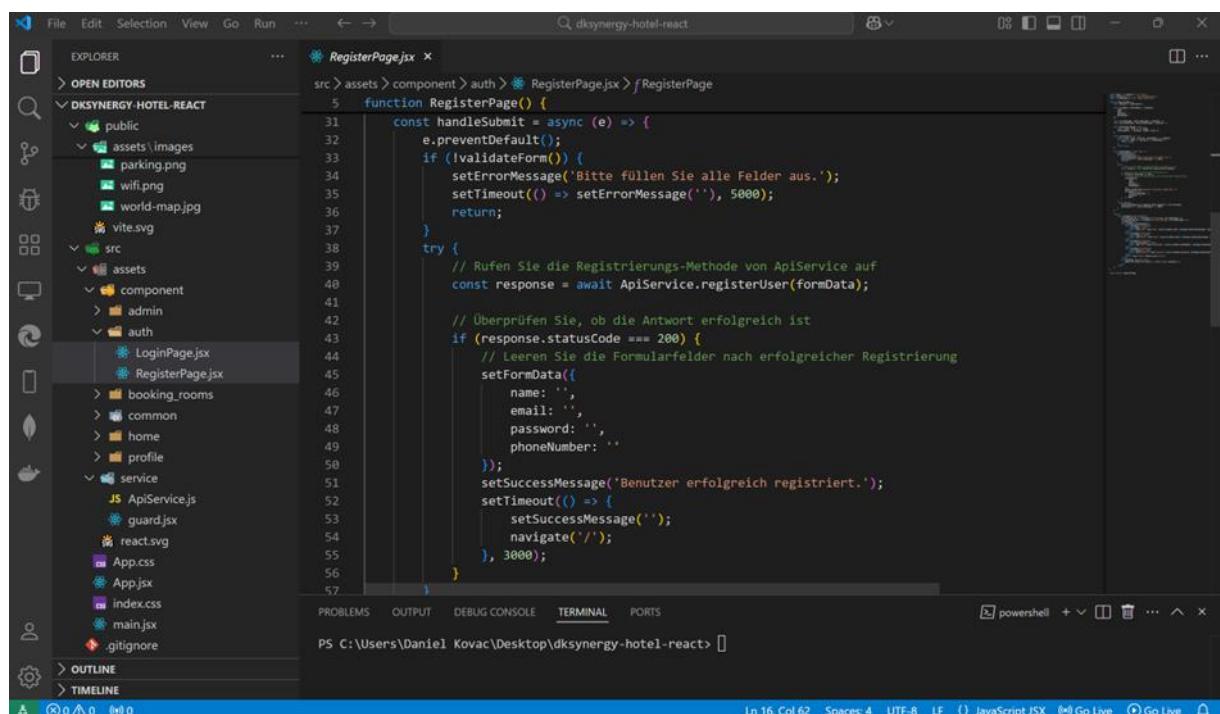
```

Registrierungsseite (RegisterPage.jsx) erstellen

Die RegisterPage-Komponente bildet die Grundlage für die Nutzerregistrierung im Hotelbuchungssystem. Sie implementiert ein validiertes Anmeldeformular, das die essentiellen Benutzerdaten erfasst. Die Komponente verarbeitet die Eingabefelder für Name, E-Mail, Telefonnummer und Passwort, wobei eine integrierte Validierungslogik die Vollständigkeit aller Eingaben sicherstellt.

Der Registrierungsprozess läuft über eine API-Anbindung, die bei erfolgreicher Registrierung eine automatische Weiterleitung zur Login-Seite initiiert. Das System bietet unmittelbares Feedback durch temporäre Benachrichtigungen: Fehlermeldungen bei unvollständigen Eingaben werden für 5 Sekunden angezeigt, während Erfolgsmeldungen nach 3 Sekunden zur Login-Weiterleitung führen.

Die Implementierung gewährleistet dabei die sichere Verarbeitung sensibler Daten, indem Passwörter ausschliesslich über die API-Schnittstelle übermittelt und nie lokal gespeichert werden. Dieser fokussierte Ansatz optimiert die Benutzererfahrung bei gleichzeitiger Gewährleistung der Datensicherheit.



The screenshot shows a code editor interface with the following details:

- File Path:** src > assets > component > auth > RegisterPage.jsx
- Code Snippet:**

```

function RegisterPage() {
  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!validateForm()) {
      setErrorMessage('Bitte füllen Sie alle Felder aus.');
      setTimeout(() => setErrorMessage(''), 5000);
      return;
    }
    try {
      // Rufen Sie die Registrierungs-Methode von ApiService auf
      const response = await ApiService.registerUser(formData);

      // Überprüfen Sie, ob die Antwort erfolgreich ist
      if (response.statusCode === 200) {
        // Leeren Sie die Formularfelder nach erfolgreicher Registrierung
        setFormData({
          name: '',
          email: '',
          password: '',
          phoneNumber: ''
        });
        setSuccessMessage('Benutzer erfolgreich registriert.');
        setTimeout(() => {
          setSuccessMessage('');
          navigate('/');
        }, 3000);
      }
    }
  }
}

```
- IDE Interface:** The editor includes standard UI elements like File, Edit, Selection, View, Go, Run, etc. The bottom status bar shows the file path as "C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react>" and other system information.

Benutzerprofilseite (ProfilePage.jsx) erstellen

Diese zentrale Komponente ermöglicht eine effiziente Verwaltung des Benutzerkontos und bietet schnellen Zugriff auf alle relevanten Buchungsinformationen.

Hauptfunktionen

- Anzeige der Benutzerinformationen (Name, E-Mail, Telefon)
- Verwaltung der Buchungshistorie
- Profil-Bearbeitungsmöglichkeiten
- Abmelde-Funktion

Technische Implementierung

1. Automatischer Datenabruf

- Laden der Profildaten via ApiService.getUserProfile()
- Abrufen der Buchungshistorie über ApiService.getUserBookings()
- Fehlerbehandlung bei API-Ausfällen

2. Buchungsanzeige

- Chronologische Auflistung aller Buchungen
- Pro Buchung: Code, Daten, Gästeanzahl, Zimmerdetails
- Fallback-Anzeige bei fehlenden Buchungen

3. Benutzerinteraktionen

- Navigation zur Profilbearbeitung
- Sicherer Logout-Prozess
- Token-Entfernung bei Abmeldung

Fehlerbehandlung

- API-Fehlerüberwachung
- Benutzerbenachrichtigungen
- Ladezustandsanzeige

Profilbearbeitungsseite (EditProfilePage.jsx) erstellen

Die Profilbearbeitungsseite wurde als zentrale Komponente für die Verwaltung von Benutzerkonten implementiert. Sie ermöglicht Nutzern die Einsicht und Bearbeitung ihrer persönlichen Daten sowie die Option zur Kontoentfernung.

Die Komponente basiert auf einer effizienten State-Management-Struktur und kommuniziert direkt mit dem ApiService für alle datenbezogenen Operationen. Beim initialen Laden der Seite werden automatisch die aktuellen Benutzerdaten abgerufen und angezeigt. Dies umfasst:

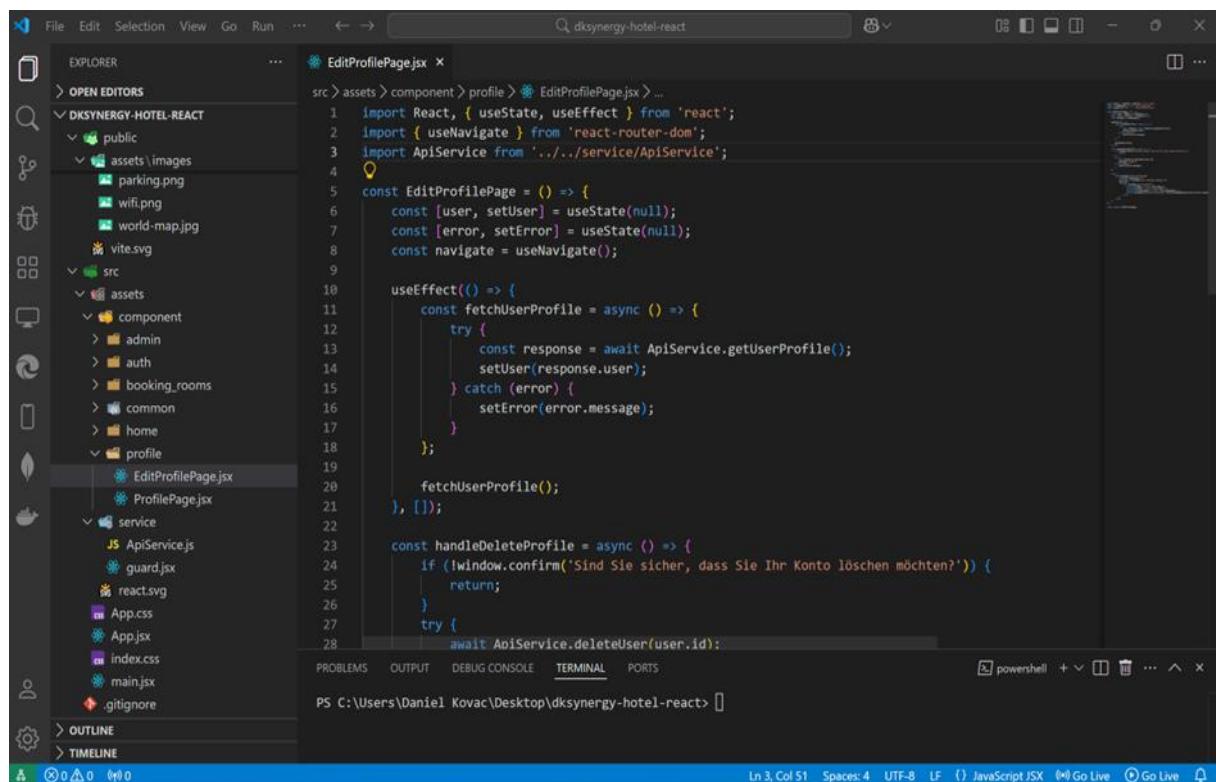
- Persönliche Informationen (Name, E-Mail, Telefonnummer)
- Kontoverwaltungsoptionen
- Sicherheitseinstellungen

Ein besonderer Fokus liegt auf der sicheren Handhabung kritischer Operationen. Die Kontolöschfunktion ist durch einen mehrstufigen Bestätigungsprozess geschützt:

1. Initialer Löschauftrag durch Benutzer
2. Anzeige eines Bestätigungsdialogs
3. Finale Bestätigung vor der Durchführung

Die Komponente implementiert ebenfalls ein umfassendes Fehlermanagementsystem:

- Sofortige Validierung von Benutzereingaben
- Klare Fehlerbenachrichtigungen bei API-Kommunikationsproblemen
- Automatische Weiterleitung nach erfolgreichen Aktionen



```

File Edit Selection View Go Run ... ← → dksynergy-hotel-react
EXPLORER OPEN EDITORS DKSYNERGY-HOTEL-REACT
public assets\images parking.png wifi.png world-map.jpg vite.svg
src assets component admin auth booking_rooms common home profile
EditProfilePage.jsx ProfilePage.jsx
service ApiService.js guard.jsx react.svg App.css App.jsx index.css main.jsx .gitignore
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + powershell
PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react>

```

```

EditProfilePage.jsx
src > assets > component > profile > EditProfilePage.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import ApiService from '../../../../../service/ApiService';
4
5 const EditProfilePage = () => {
6   const [user, setUser] = useState(null);
7   const [error, setError] = useState(null);
8   const navigate = useNavigate();
9
10  useEffect(() => {
11    const fetchUserProfile = async () => {
12      try {
13        const response = await ApiService.getUserProfile();
14        setUser(response.user);
15      } catch (error) {
16        setError(error.message);
17      }
18    };
19
20    fetchUserProfile();
21  }, []);
22
23  const handleDeleteProfile = async () => {
24    if (!window.confirm('Sind Sie sicher, dass Sie Ihr Konto löschen möchten?')) {
25      return;
26    }
27    try {
28      await ApiService.deleteUser(user.id);
29    } catch (error) {
30      setError(error.message);
31    }
32  };
33
34  const handleLogout = () => {
35    localStorage.removeItem('user');
36    navigate('/login');
37  };
38
39  return (
40    <div>
41      <h1>Edit Profile</h1>
42      <p>Name: {user.name}</p>
43      <p>Email: {user.email}</p>
44      <p>Phone: {user.phone}</p>
45      <button onClick={handleDeleteProfile}>Delete Profile</button>
46      <button onClick={handleLogout}>Logout</button>
47    </div>
48  );
49}

```

Administration-Dashboard-Seite (AdminPage.jsx) erstellen

Das Administrations-Dashboard wurde als zentrale Steuerungseinheit für Systemadministratoren entwickelt und dient als Hauptzugang zu allen Verwaltungsfunktionen des Hotelsystems.

Der Dashboard bietet folgende Kernfunktionen:

- Automatischer Abruf und Anzeige der Administrator-Profildata
- Personalisierte Begrüßung mit Namen des eingeloggten Administrators
- Direkter Zugriff auf die Zimmerverwaltung via "/admin/manage-rooms"
- Zentraler Zugang zum Buchungsmanagement über "/admin/manage-bookings"

Die technische Umsetzung beinhaltet:

- Integration des ApiService für Profildata-Abruf
- Implementierung von React useEffect für automatisches Laden
- Rollenbasierte Zugangskontrolle für Administratoren
- Sichere Routing-Struktur für administrative Funktionen

Diese Implementation gewährleistet eine effiziente und sichere Verwaltung aller Hotelsystemfunktionen von einer zentralen Stelle aus. Das Dashboard vereinfacht dabei die tägliche Arbeit der Administratoren durch seine intuitive Struktur und direkten Zugriff auf alle wichtigen Verwaltungsfunktionen.

Zimmermanagement-Seite (ManageRoomPage.jsx) erstellen

Die ManageRoomPage dient als zentrale Verwaltungsoberfläche für das Zimmermanagement des Hotels. Sie bietet Administratoren folgende Hauptfunktionen:

1. Datenverwaltung und Anzeige

- Automatischer Abruf und Anzeige aller Zimmer beim Seitenaufruf
- Speicherung der Daten in separaten States für Originalansicht und gefilterte Ansicht
- Integration einer Paginierung für übersichtliche Darstellung

2. Filterfunktionen

- Dynamischer Abruf verfügbarer Zimmertypen
- Filterung der Zimmeransicht nach spezifischen Kategorien
- Sofortige Aktualisierung der gefilterten Ansicht

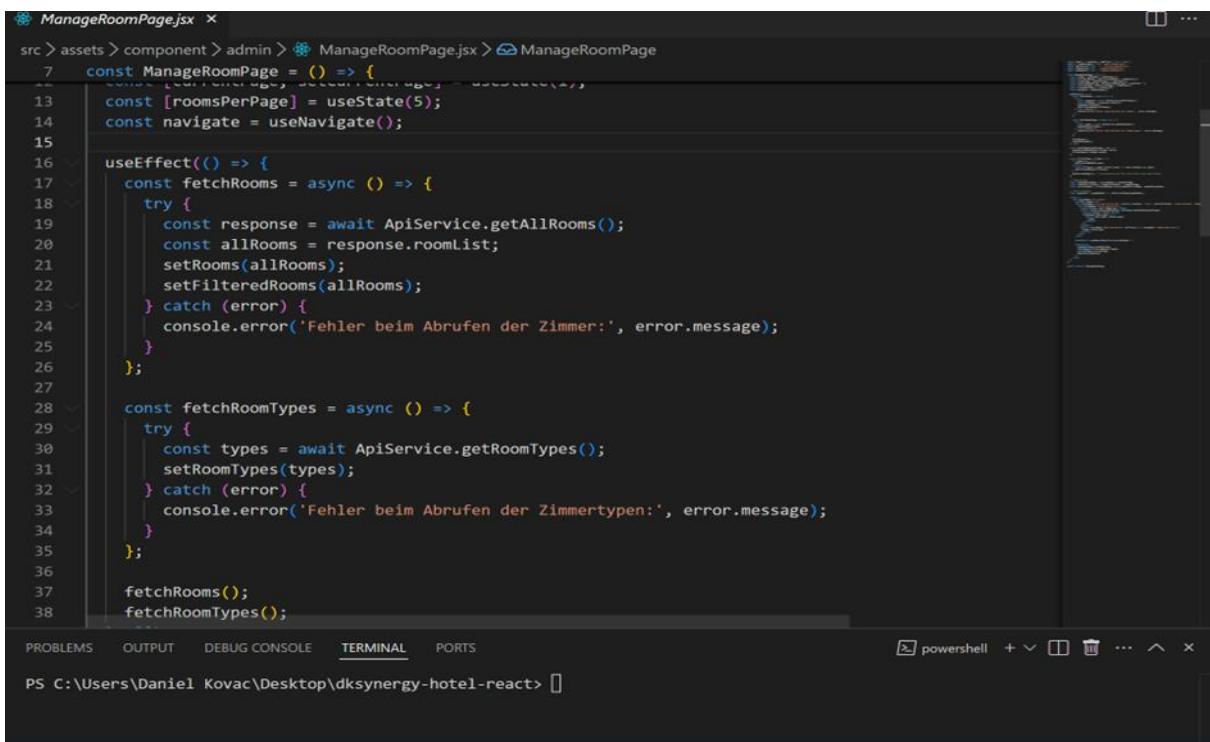
3. Administrative Funktionen

- Direkte Navigation zur Zimmerhinzufügung
- Bearbeitungsmöglichkeiten bestehender Zimmer
- Löschfunktion für nicht mehr benötigte Zimmer

4. Benutzerinterface

- RoomResult-Komponente zur Darstellung der Zimmerinformationen
- Navigationselemente für Seitenwechsel
- Intuitive Bedienelemente für Verwaltungsaktionen

Diese zentrale Verwaltungskomponente optimiert den Administrationsprozess durch ihre übersichtliche Struktur und effiziente Funktionalität.



```

ManageRoomPage.jsx
src > assets > component > admin > ManageRoomPage.jsx > ManageRoomPage

7  const ManageRoomPage = () => {
8    const [rooms, setRooms] = useState([]);
9    const [filteredRooms, setFilteredRooms] = useState([]);
10   const [roomTypes, setRoomTypes] = useState([]);
11   const [roomsPerPage] = useState(5);
12   const [currentPage, setCurrentPage] = useState(1);
13   const [totalPages, setTotalPages] = useState(1);
14   const navigate = useNavigate();
15
16   useEffect(() => {
17     const fetchRooms = async () => {
18       try {
19         const response = await ApiService.getAllRooms();
20         const allRooms = response.roomList;
21         setRooms(allRooms);
22         setFilteredRooms(allRooms);
23       } catch (error) {
24         console.error('Fehler beim Abrufen der Zimmer:', error.message);
25       }
26     };
27
28     const fetchRoomTypes = async () => {
29       try {
30         const types = await ApiService.getRoomTypes();
31         setRoomTypes(types);
32       } catch (error) {
33         console.error('Fehler beim Abrufen der Zimmertypen:', error.message);
34       }
35     };
36
37     fetchRooms();
38     fetchRoomTypes();
  
```

The screenshot shows the code editor interface with the file 'ManageRoomPage.jsx' open. The code itself is a functional component that handles room data and types. It uses several state management hooks from the 'useState' and 'useEffect' families. The component performs two main asynchronous operations: fetching all rooms and fetching room types. It also manages pagination and navigation. The code is annotated with line numbers and includes explanatory comments for each section of logic.

Buchungsverwaltungsseite (ManageBookingsPage.jsx) erstellen

Die ManageBookingsPage dient als zentrale Verwaltungsplattform für alle Hotelbuchungen. Sie bietet Administratoren folgende wesentliche Funktionen:

1. Buchungsverwaltung und Anzeige

- Automatischer Abruf aller Buchungen beim Seitenaufruf via ApiService
- Duale Datenspeicherung für Original- und gefilterte Ansichten
- Integrierte Paginierung mit 6 Buchungen pro Seite

2. Such- und Filterfunktionen

- Direkte Suche nach spezifischen Buchungsnummern
- Echtzeit-Filterung der angezeigten Buchungen
- Automatische Aktualisierung der gefilterten Ansicht

3. Verwaltungsfunktionen

- Detailansicht jeder Buchung
- Direkter Zugriff auf Bearbeitungsmöglichkeiten
- Navigation zur EditBookingPage für Buchungsmodifikationen

Die Buchungsverwaltungsseite optimiert den administrativen Workflow durch ihre effiziente Organisation und benutzerfreundliche Struktur. Die implementierte Paginierung und Suchfunktion ermöglichen auch bei hohem Buchungsaufkommen eine übersichtliche und effektive Verwaltung.

```

  ManageBookingsPage.jsx
  src > assets > component > admin > ManageBookingsPage.jsx > ManageBookingsPage
  6  const ManageBookingsPage = () => {
  52
  53    const paginate = (pageNumber) => setCurrentPage(pageNumber);
  54
  55    return (
  56      <div className='bookings-container'>
  57        <h2>Alle Buchungen</h2>
  58        <div className='search-div'>
  59          <label>Filtern nach Buchungsnummer:</label>
  60          <input
  61            type="text"
  62            value={searchTerm}
  63            onChange={handleSearchChange}
  64            placeholder="Buchungsnummer eingeben"
  65          />
  66        </div>
  67
  68        <div className="booking-results">
  69          {currentBookings.map((booking) => (
  70            <div key={booking.id} className="booking-result-item">
  71              <p><strong>Buchungscode:</strong> {booking.bookingConfirmationCode}</p>
  72              <p><strong>Check-in Datum:</strong> {booking.checkInDate}</p>
  73              <p><strong>Check-out Datum:</strong> {booking.checkOutDate}</p>
  74              <p><strong>Gesamtanzahl der Gäste:</strong> {booking.totalNumOfGuest}</p>
  75              <button
  76                className="edit-room-button"
  77                onClick={() => navigate(`/admin/edit-booking/${booking.bookingConfirmationCode}`)}
  78              >Buchung verwalten</button>
  
```

The screenshot shows the code editor interface with the ManageBookingsPage.jsx file open. The code defines a functional component that handles pagination and displays booking results with a search bar and edit button. The code uses React components like <div>, <h2>, <input>, and <button>. It also uses hooks like useState and useEffect. The code is well-structured with clear comments and variable names.

Zimmerhinzufügungsseite (AddRoomPage.jsx) erstellen

Die AddRoomPage bildet die zentrale Komponente zur Erweiterung des Zimmerbestands im Hotelsystem. Sie ermöglicht Administratoren eine strukturierte Erfassung neuer Zimmer mit allen relevanten Informationen.

1. Grundfunktionen und Datenerfassung

- Bildupload mit direkter Vorschaufunktion
- Preisgestaltung und Zimmerbeschreibung
- Flexible Zimmertyp-Zuweisung
- Automatische Datenvielfältigung

2. Dynamische Zimmertyp-Verwaltung

- Automatischer Abruf existierender Zimmertypen
- Option zur Erstellung neuer Kategorien
- Intelligente Dropdown-Integration

3. Speicherprozess und Sicherheit

- Mehrstufige Formularvalidierung
- Bestätigungsdialog vor finalem Speichern
- Automatische Weiterleitung nach erfolgreicher Erstellung

Die Komponente gewährleistet durch ihre intuitive Struktur und umfassende Validierung eine effiziente und fehlerfreie Zimmerverwaltung. Die implementierte Vorschaufunktion und flexiblen Eingabeoptionen optimieren den administrativen Workflow bei der Zimmererfassung.

Buchungsbearbeitungsseite (EditBookingPage.jsx) erstellen

Die EditBookingPage fungiert als administrative Schnittstelle zur Verwaltung einzelner Hotelbuchungen. Sie dient der detaillierten Ansicht und dem Abschluss von Buchungen im System.

1. Kernfunktionen

- Automatischer Abruf spezifischer Buchungsdetails via ApiService
- Umfassende Darstellung aller buchungsrelevanten Informationen
- Integrierte Buchungsabschlussfunktion mit Sicherheitsabfrage

2. Informationsanzeige

- Buchungscode und Zeiträume
- Gästedetails und Kontaktinformationen
- Zimmerspezifikationen inkl. Bildmaterial
- Preisübersicht und Buchungsstatus

3. Prozesssteuerung

- Validierte Buchungsabschlussprozesse
- Automatisierte Erfolgs- und Fehlermeldungen
- Gezielte Weiterleitung nach Aktionsabschluss

Die Komponente optimiert den administrativen Workflow durch ihre übersichtliche Darstellung und effiziente Prozesssteuerung. Integrierte Sicherheitsabfragen und Feedback-Mechanismen gewährleisten dabei eine zuverlässige Buchungsverwaltung.

Zimmerbearbeitungsseite (EditRoomPage.jsx) erstellen

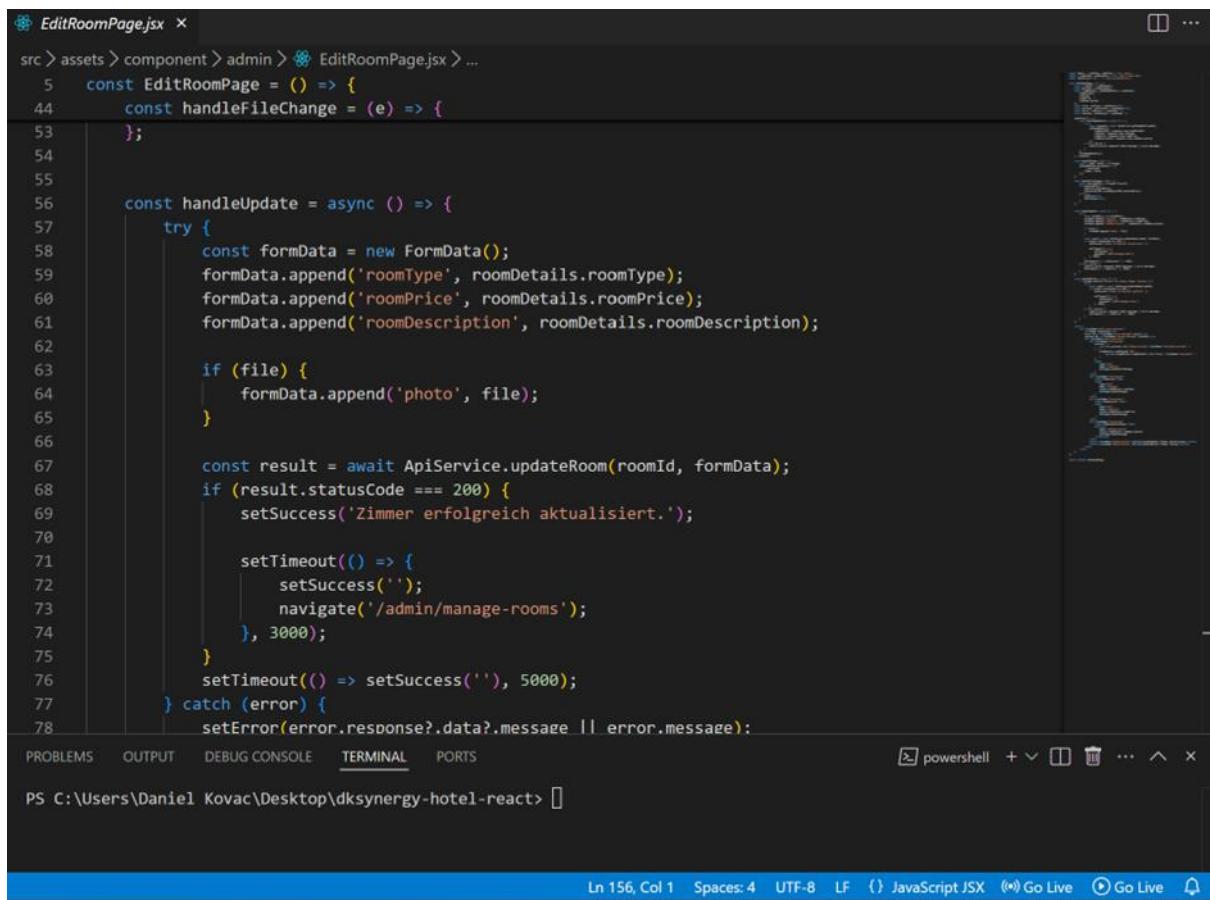
Die EditRoomPage dient Administratoren zur Verwaltung existierender Zimmer im Hotelsystem. Bei Aufruf der Seite werden automatisch die aktuellen Zimmerdaten via ApiService.getRoomById(roomId) geladen und in einem bearbeitbaren Formular dargestellt.

Administratoren können folgende Aktionen durchführen:

- Anpassung von Zimmertyp, Preis und Beschreibung
- Upload neuer Zimmerbilder mit integrierter Vorschau
- Komplette Löschung eines Zimmers aus dem System

Die Datenverarbeitung erfolgt über FormData-Objekte, die alle aktualisierten Informationen sammeln und via ApiService.updateRoom() an das Backend übermitteln. Bei erfolgreicher Aktualisierung erscheint eine Bestätigungsmeldung, gefolgt von einer automatischen Weiterleitung zur Zimmerverwaltung nach 3 Sekunden.

Der Löschprozess ist durch einen Bestätigungsdialog abgesichert. Nach erfolgreicher Löschung erfolgt die Weiterleitung zum Admin-Dashboard. Sowohl Erfolgs- als auch Fehlermeldungen werden temporär angezeigt (3-5 Sekunden) und bieten direkte Rückmeldung über den Prozessstatus.



```

EditRoomPage.jsx ×

src > assets > component > admin > EditRoomPage.jsx > ...
5  const EditRoomPage = () => {
44    const handleFileChange = (e) => {
53      };
54
55
56      const handleUpdate = async () => {
57        try {
58          const formData = new FormData();
59          formData.append('roomType', roomDetails.roomType);
60          formData.append('roomPrice', roomDetails.roomPrice);
61          formData.append('roomDescription', roomDetails.roomDescription);
62
63          if (file) {
64            formData.append('photo', file);
65          }
66
67          const result = await ApiService.updateRoom(roomId, formData);
68          if (result.statusCode === 200) {
69            setSuccess('Zimmer erfolgreich aktualisiert.');
70
71            setTimeout(() => {
72              setSuccess('');
73              navigate('/admin/manage-rooms');
74            }, 3000);
75          }
76          setTimeout(() => setSuccess(''), 5000);
77        } catch (error) {
78          setError(error.response?.data?.message || error.message);
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> []

Ln 156, Col 1 Spaces: 4 UTF-8 LF {} JavaScript JSX Go Live Go Live

Routing- und Seitenstruktur (App.jsx) verbessern

Die App.jsx wurde um zusätzliche Funktionen und eine verfeinerte Routingstruktur erweitert, um die Anwendungssicherheit und Benutzerführung zu optimieren.

Routing-Struktur

1. Öffentlicher Bereich

- Startseite (/home)
- Authentifizierung (/login, /register)
- Zimmerübersicht (/rooms)
- Buchungssuche (/find-booking)

2. Geschützter Bereich

- Zimmerbuchung (/room-details-book/:roomId)
- Profilverwaltung (/profile, /edit-profile)

3. Administrativer Bereich

- Dashboard (/admin)
- Zimmermanagement (/admin/manage-rooms, /admin/edit-room/:roomId, /admin/add-room)
- Buchungsverwaltung (/admin/manage-bookings, /admin/edit-booking/:bookingCode)

Sicherheitsimplementierung

Die Zugriffskontrolle wurde durch mehrschichtige Sicherheitsmechanismen verstärkt:

- Rollenbasierte Routenführung
- Authentifizierungsprüfung bei geschützten Routen
- Automatische Weiterleitung bei fehlenden Berechtigungen

UI-Integration

Die Anwendung behält eine konsistente Struktur durch:

- Persistente Navigationsleiste
- Einheitlichen Footer
- Kontextabhängige Menüführung

Diese Erweiterungen optimieren die Benutzererfahrung und gewährleisten gleichzeitig eine sichere Anwendungsumgebung.

Endresultat

Das Frontend des Hotelbuchungssystems wurde erfolgreich mit React und Vite umgesetzt. Das System läuft auf dem lokalen Entwicklungsserver unter <http://localhost:5173> und bietet eine vollständig funktionsfähige Benutzeroberfläche.

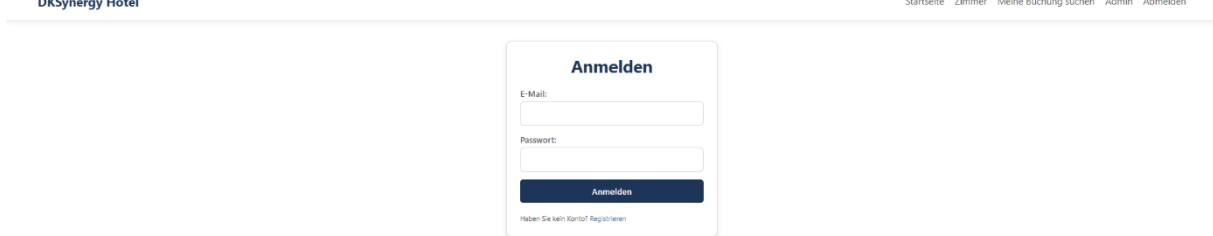
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

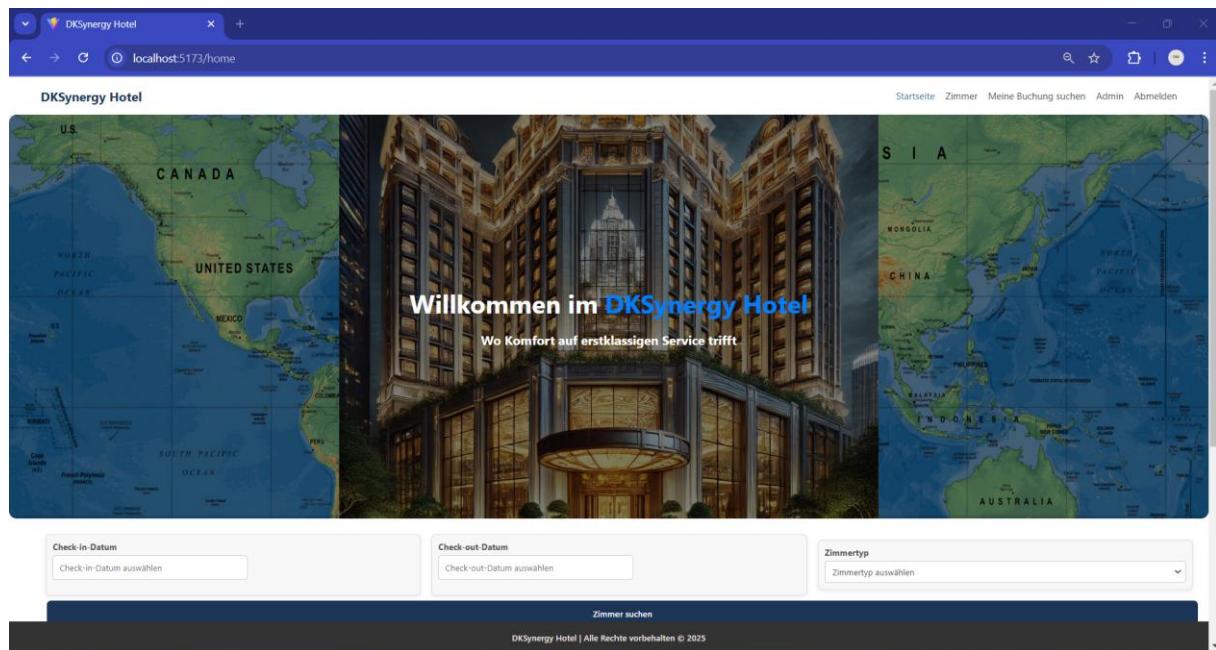
Files in the public directory are served at the root path.
PS C:\Users\Daniel Kovac\Desktop\dksynergy-hotel-react> npm run dev

> dksynergy-hotel-react@0.0.0 dev
> vite --config vite.config.js

VITE v6.0.11 ready in 153 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```





Die finale Version bietet:

- Benutzerfreundliche Navigation
- Rollenbasierte Zugriffssteuerung
- Dynamische Zimmerverwaltung
- Integriertes Buchungssystem

Das System wurde ausgiebig getestet und läuft stabil. Die Implementierung erfüllt alle definierten Anforderungen und bietet eine solide Basis für zukünftige Erweiterungen.

Frontend testen

Testfall 1: Registrierung eines neuen Benutzers

Ein vollständiger Registrierungs- und Login-Zyklus wurde mit folgenden Testdaten durchgeführt:

- Name: Maria Musterfrau
- E-Mail: maria.musterin@gmail.com
- Telefonnummer: 0791111111
- Passwort: [Sicher gespeichert]

Validierungstests

Die Sicherheitsmechanismen wurden durch verschiedene Szenarien geprüft:

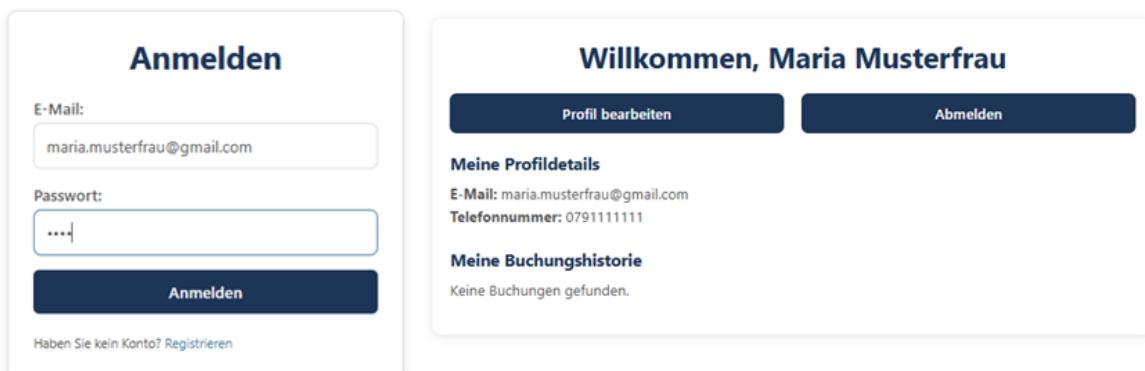
- Eingabe ungültiger E-Mail-Formate
- Tests mit zu schwachen Passwörtern
- Versuche mit fehlenden Pflichtfeldern
- Duplikat-Registrierungsversuche

Testergebnisse

Die Implementierung zeigte durchweg positives Verhalten:

- Erfolgreiche Registrierung mit korrekter Bestätigungsmeldung
- Reibungslose Weiterleitung zum Login-Bereich
- Korrekte Profildarstellung nach Anmeldung
- Angemessene Fehlerbehandlung bei ungültigen Eingaben

Die Tests bestätigen die robuste und benutzerfreundliche Implementierung des Registrierungs- und Anmeldeprozesses, der sowohl Sicherheitsanforderungen als auch Benutzerfreundlichkeit erfolgreich vereint.



Testfall 2: Hinzufügen eines neuen Zimmers als Admin

Die Funktionalität zum Hinzufügen neuer Zimmer wurde umfassend getestet, wobei ein Admin-Account für die Durchführung der Tests verwendet wurde.



Ein neues Zimmer wurde mit folgenden Testdaten erfolgreich angelegt:

- **Zimmertyp:** Einzelzimmer
- **Preis:** CHF 500
- **Beschreibung:** "Unser Märchen- und Fantasie-Zimmer vereint zauberhafte Details mit luxuriösem Komfort für ein unvergessliches Erlebnis."
- **Bildmaterial:** Erfolgreich hochgeladen und angezeigt

Verschiedene Fehlerfälle wurden überprüft:

- Formularübermittlung mit leeren Pflichtfeldern
- Eingabe ungültiger Preisangaben
- Zimmeranlage ohne Bildupload
- Duplikierung existierender Zimmertypen

Die implementierte Funktionalität zeigt folgendes Verhalten:

- Erfolgreiche Zimmeranlage mit Bestätigungsmeldung
- Korrekte Weiterleitung zur Zimmerverwaltung
- Sofortige Sichtbarkeit des neuen Zimmers in der Übersicht
- Angemessene Fehlerbehandlung bei ungültigen Eingaben

Die Tests bestätigen eine robuste Implementierung der Zimmer-Administrationsfunktionen, die sowohl benutzerfreundlich als auch fehlertolerant arbeitet.

The screenshot displays two cards for room types. The first card is for "Einzelzimmer" (Single Room) with a price of €500 per night and a description about a magical and fantastical theme. The second card is for "Modernes Einzelzimmer" (Modern Single Room) with a price of €200 per night and a description about a modern single room. Both cards include a "Raum bearbeiten" (Edit Room) button.

Frontend-Architektur

Das Frontend des DKSynergy Hotels basiert auf einer modernen React-Anwendung mit Vite als Build-Tool. Diese Kombination wurde gewählt, nachdem Kompatibilitätsprobleme mit React 19 auftraten. Der Stack umfasst:

- React + Vite für effiziente Entwicklung und schnelle Build-Zeiten
- React Router DOM für clientseitiges Routing
- Axios für API-Kommunikation
- React DatePicker für Buchungsfunktionalitäten

Architekturelle Schichten

Die Anwendung ist in klar definierte Schichten unterteilt:

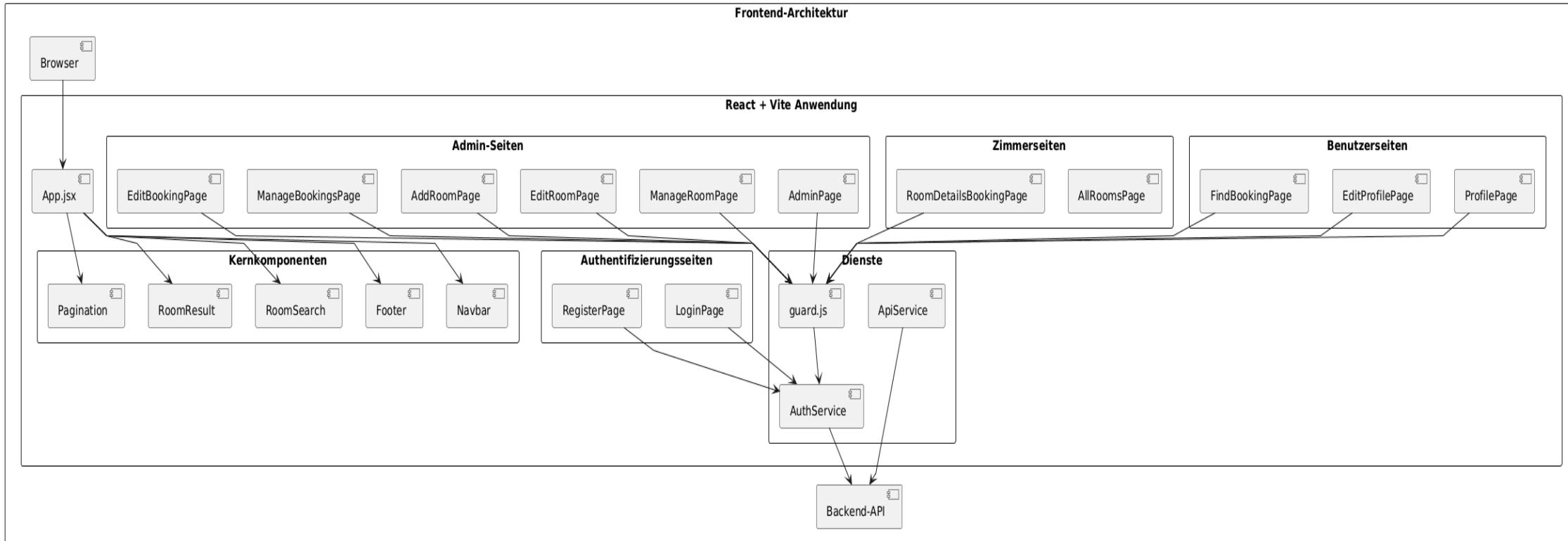
- **Core Components:** Wiederverwendbare UI-Komponenten wie Navbar, Footer, RoomSearch
- **Pages:** Haupt-Seitenkomponenten, unterteilt in öffentliche, geschützte und administrative Bereiche
- **Services:** Zentrale Dienste für API-Kommunikation und Authentifizierung
- **Guards:** Zugriffsschutz für geschützte Routen

Sicherheitskonzept des Frontend

Die Sicherheitsarchitektur basiert auf einem mehrschichtigen Ansatz:

- Token-basierte Authentifizierung
- Rollenbasierte Zugriffskontrolle (RBAC)
- Guard-Komponenten für Routenschutz
- Sichere State-Verwaltung im localStorage

Diese Architektur ermöglicht eine klare Trennung der Verantwortlichkeiten und gewährleistet gleichzeitig eine hohe Wartbarkeit und Erweiterbarkeit des Systems.



Testprotokoll Backend und Frontend

Testfall	Erwartetes Ergebnis	Tatsächliches Ergebnis	Bestanden
Frontend			
1. Registrierung neuer Benutzer	Benutzer kann sich erfolgreich registrieren und erhält Bestätigung, Weiterleitung zum Login	Registrierung erfolgreich, korrekte Weiterleitung, Fehler werden abgefangen	Ja
2. Hinzufügen eines neuen Zimmers (Admin)	Neues Zimmer (z.B. Einzelzimmer) wird angelegt, Bestätigungsmeldung erscheint sofort, Fehlermeldungen bei ungültigen Daten	Zimmer korrekt erstellt, Bild hochgeladen, Fehlerhandlung bei falschen Angaben	Ja
Backend			
1. User erstellen	User (z.B. „Max“) mit gehashtem Passwort in DB gespeichert, Rolle funktioniert	User erfolgreich in DB, Rolle ADMIN und USER funktional	Ja
2. User einloggen	Login erzeugt gültiges JWT-Token (7 Tage)	Token wurde erstellt, Login erfolgreich	Ja
3. Alle User abrufen	Nur ADMIN kann /users/all aufrufen, USER erhält Fehlermeldung	ADMIN sieht Userliste, USER bekommt erwartete Fehlermeldung	Ja
4. Bookings	/get-user-bookings/{userId} gibt leere Liste zurück, falls keine Buchungen vorhanden sind	Korrekte Ausgabe: leere Liste bei fehlenden Buchungen	Ja
5a. User löschen (Fehlerfall)	Löschen eines nichtexistierenden Benutzers führt zu Meldung „Benutzer nicht gefunden“	404 und Fehlermeldung korrekt zurückgegeben	Ja
5b. User löschen (Erfolgfall)	Löschen eines existierenden Benutzers liefert Erfolgsmeldung, DB wird aktualisiert	Status 200, User ist nicht mehr in der DB, andere User unverändert	Ja
6. Zimmer hinzufügen	Zimmerdaten und Bild werden korrekt gespeichert, Fehlerbehandlung bei unvollständigen Daten	Zimmer mit Bild (AWS S3) erfolgreich angelegt, bei fehlerhaften Eingaben gab's korrekte Fehlermeldungen	Ja

Sicherheitskonzept

Damit unser Hotelbuchungssystem sicher ist und keine unbefugten Personen auf sensible Daten zugreifen können, haben wir verschiedene Schutzmaßnahmen eingebaut.

Authentifizierung & Autorisierung

- **Sicheres Einloggen mit JWT**

Wir verwenden "JSON Web Tokens" (JWT), um sicherzustellen, dass nur berechtigte Nutzer auf bestimmte Bereiche der Anwendung zugreifen können.

- **Sichere Passwörter**

- **Passwörter werden nicht im Klartext gespeichert**, sondern mit Hashing und Salting geschützt.
- **Sicherheitsregeln** sorgen dafür, dass Nutzer starke Passwörter verwenden.
- **Token-Speicherung**: Sichere Speicherung, um Missbrauch zu vermeiden.

Datensicherheit

- **Schutz der Server-Anfragen**

- Bestimmte Bereiche der Webseite sind nur für eingeloggte Benutzer oder Admins sichtbar.
- **Eingaben werden überprüft**, um Manipulationen oder Schadcode zu verhindern.

- **Sicherer Umgang mit Bildern**

- Die Hotelbilder werden sicher auf Amazon Web Services (AWS) gespeichert.
- **Zugriffsrechte sind streng geregelt**, sodass nur die nötigen Personen Änderungen vornehmen können.

Schutz vor Angriffen

- **Schutz vor Schadcode**: Alle Daten, die ins System kommen, werden vorher überprüft.
- **Keine direkten Datenbank-Zugriffe**: Damit verhindern wir Manipulationen oder Datendiebstahl.

API-Sicherheit

- **Rate-Limiting**: Begrenzung der Anfragen, um Überlastung und Angriffe zu verhindern.
- **Secure Headers**: Schutzmaßnahmen wie CORS und Content Security Policy (CSP) sind aktiv.
- **Zugriffsschutz für API-Endpunkte**: Nur authentifizierte Nutzer haben Zugriff.

Frontend-Sicherheit

- **XSS-Schutz durch React**: React filtert schädlichen Code automatisch heraus.
- **Geschützte Routen (ProtectedRoute)**: Nutzer ohne Berechtigung können nicht auf geschützte Seiten zugreifen.
- **Validierung von Benutzereingaben**: Unsichere Eingaben werden erkannt und blockiert.

Abläufe beim Login

Der Login-Prozess im Hotelbuchungssystem gewährleistet eine sichere Authentifizierung durch JWT.

1. Login-Daten eingeben

- Benutzer gibt E-Mail & Passwort in **LoginPage.jsx** ein.
- Leere Felder lösen eine Fehlermeldung aus.

2. Authentifizierungsanfrage an Backend

- **POST-Anfrage** an /auth/login mit E-Mail & Passwort.

3. Validierung im Backend

- **UserService** prüft E-Mail.
- Passwortabgleich mit **BCryptPasswordEncoder**.
- JWT-Token wird generiert oder Fehler zurückgegeben.

4. Token-Generierung & Antwort

- **JWTUtils** erstellt Token mit Benutzerrolle (ADMIN/USER).
- Antwort enthält Token & Erfolgsmeldung.

5. Speicherung im Frontend

- Token & Benutzerrolle werden im **localStorage** gesichert.

6. Weiterleitung

- **USER:** Startseite/Profil
- **ADMIN:** Admin-Dashboard

7. Token-Nutzung für geschützte Anfragen

- JWT wird im **Authorization-Header** übermittelt.
- **JWTAuthFilter** prüft Token auf Gültigkeit.

8. Logout

- Token wird aus **localStorage** entfernt.
- Weiterleitung zur Login-Seite.

Diese Struktur sorgt für eine sichere und effiziente Authentifizierung.

Arbeitsjournal

Daniel Kovac

Datum	Dezember 2024
Zeit	ca. 6-7 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Einrichtung des Spring Boot Projekts mit Maven. ○ Konfiguration der Datenbankverbindung mit PostgreSQL/MongoDB. ○ Implementierung der Benutzerregistrierung und -anmeldung mit JWT-Authentifizierung. ○ Erstellung der UserService-Klasse mit folgenden Funktionen: <ul style="list-style-type: none"> ▪ Registrierung neuer Benutzer. ▪ Benutzer-Login mit Token-Generierung. ▪ Validierung von Passwörtern & Benutzerrollen. ○ Einführung der JWT-Security-Konfiguration mit Spring Security. ○ Erstes Testen der API mit Postman.
Nächste Schritte	<ul style="list-style-type: none"> ○ Fertigstellung der Token-Validierung und Rollenbasierten Zugriffskontrolle (RBAC). ○ Erstellung des Room-Management-Systems im Backend.

Datum	25.01.2025
Zeit	ca. 7-8 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Implementierung des RoomService für die Zimmerverwaltung: <ul style="list-style-type: none"> ▪ Hinzufügen, Bearbeiten, Löschen von Zimmern. ▪ Abrufen aller verfügbaren Zimmer. ▪ Filtern nach Zimmertyp. ○ Einrichtung des AWS S3 Buckets zur Speicherung von Zimmerbildern. ○ Erstellung der AWS-S3-Service-Klasse für den Upload von Bildern. ○ Erster Test des Uploads mit Postman – Fehler 403 (Access Denied) erhalten. ○ Debugging des IAM-Rollenproblems in AWS. ○ Anpassung der S3 Bucket-Policy, um Uploads zu ermöglichen.
Nächste Schritte	<ul style="list-style-type: none"> ○ Fertigstellung der Zimmerverwaltung im Backend. ○ Entwicklung der Buchungslogik mit Datenbankintegration.

Datum	26.01.2025
Zeit	ca. 8-9 Stunden

Tätigkeiten	<ul style="list-style-type: none"> ○ Erstellung des BookingService zur Verwaltung von Buchungen: <ul style="list-style-type: none"> ▪ Erstellung neuer Buchungen mit Check-in & Check-out. ▪ Prüfung der Zimmerverfügbarkeit. ▪ Stornierung von Buchungen. ○ Einführung der getAvailableRoomsByDateAndType Methode: <ul style="list-style-type: none"> ▪ Prüfung, ob ein Zimmer im angegebenen Zeitraum verfügbar ist. ▪ Ausschluss bereits gebuchter Zimmer. ○ Validierung von Buchungen: <ul style="list-style-type: none"> ▪ Check-in darf nicht in der Vergangenheit liegen. ▪ Check-out muss nach Check-in sein. ○ Erweiterung der Benutzerverwaltung, um Buchungen zu speichern. ○ Erstellung eines einzigartigen Buchungscodes zur Identifikation. ○ Test mit Postman: Buchungen erfolgreich, aber ein Fehler mit Zeitzonen in Date-Feldern wurde entdeckt.
Nächste Schritte	<ul style="list-style-type: none"> ○ Debugging der Zeitzonenprobleme in Spring Boot. ○ Implementierung von Admin-Funktionen für Buchungen.

Datum	27.01.2025
Zeit	ca. 7 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Behebung des Zeitzonenfehlers durch Konfiguration von spring.datasource.timezone. ○ Erstellung der Admin-API für Buchungsmanagement: <ul style="list-style-type: none"> ▪ Abrufen aller Buchungen für Admins. ▪ Bearbeiten oder Löschen einer Buchung. ○ Verbesserung der Sicherheitsregeln in Spring Security: <ul style="list-style-type: none"> ▪ Nur Admins können Buchungen stornieren. ▪ Benutzer können nur ihre eigenen Buchungen sehen. ○ Erstellung von getUserBookingHistory, um Buchungen eines Benutzers anzuzeigen. ○ Optimierung der SQL-Abfragen zur Verbesserung der Performance. ○ Erste Tests mit Frontend-Requests via Postman durchgeführt.
Nächste Schritte	<ul style="list-style-type: none"> ○ Entwicklung der Frontend-Struktur mit React. ○ Anbindung der API an das Frontend.

Datum	28.01.2025
Zeit	ca. 6 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Einrichtung des Vite-Projekts mit React für das Frontend. ○ Implementierung der LoginPage und RegisterPage mit useState. ○ Verbindung des Frontends mit der API über Axios.

	<ul style="list-style-type: none"> ○ Speicherung des JWT-Tokens in localStorage. ○ Implementierung von geschützten Routen (ProtectedRoute, AdminRoute). ○ Erstes Styling mit CSS für Navigation & Buttons.
Nächste Schritte	<ul style="list-style-type: none"> ○ Erstellung der Zimmer- und Buchungsseiten im Frontend. ○ Implementierung der Suchfunktion für Zimmer.

Datum	29.01.2025
Zeit	ca. 5 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Implementierung der RoomSearch-Komponente zur Suche nach Zimmern. ○ Entwicklung von RoomResult zur Anzeige von Suchergebnissen. ○ Integration von Pagination für eine bessere Benutzerfreundlichkeit. ○ Erstellung der FindBookingPage, um Buchungen per Code zu suchen. ○ Verbesserte Fehlermeldungen mit Alerts für Benutzer. ○ Debugging von Problemen mit Datepickern in der Buchungsseite.
Nächste Schritte	<ul style="list-style-type: none"> ○ Fertigstellung der Buchungsseite mit Preisberechnung. ○ Einführung von Admin-Funktionalitäten im Frontend.

Datum	30.01.2025
Zeit	ca. 6 Stunden
Tätigkeiten	<ul style="list-style-type: none"> ○ Implementierung der RoomDetailsPage mit Buchungsmöglichkeit. ○ Berechnung des Gesamtpreises basierend auf Check-in & Check-out. ○ Implementierung der Admin-Seiten für: <ul style="list-style-type: none"> ▪ Zimmerverwaltung (Hinzufügen, Bearbeiten, Löschen). ▪ Buchungsverwaltung (Suchen, Abschliessen, Stornieren). ○ Profilverwaltung für Benutzer mit Möglichkeit zum Löschen des Kontos. ○ Endgültige Tests der gesamten Anwendung.
Nächste Schritte	<ul style="list-style-type: none"> ○ Schreiben der abschliessenden Dokumentation für das Projekt. ○ Letzte Bugfixes & Optimierungen vor dem Deployment.

Kristian Lubina

Datum	Dezember 2024
Zeit	ca. 4 Stunden
Tätigkeiten	<p>Anforderungsanalyse durchführen: Analyse der funktionalen und nicht-funktionalen Anforderungen für das Hotelbuchungssystem.</p> <p>User Story aufschreiben: Definition der User Stories, um die Bedürfnisse der Benutzer (Hotelgäste, Administratoren) klar zu erfassen.</p> <p>Use-Case-Diagramm erstellen: Diagramm zur Visualisierung der Hauptfunktionen des Systems, wie Zimmer buchen, Buchungen verwalten und Zimmer hinzufügen.</p> <p>ERM (Entity-Relationship-Model) erstellen: Modellierung der Datenbankstrukturen mit den Hauptentitäten <i>User</i>, <i>Room</i>, <i>Booking</i> und deren Beziehungen.</p> <p>Kapitel "Einleitung" und "Informieren und Planen" in der Dokumentation schreiben:</p> <ul style="list-style-type: none"> • Beschreibung der Projektziele, Motivation und Technologie-Stack. • Planung der Architektur und Entwicklungsphasen.
Nächste Schritte	Realisierung des Backends dokumentieren

Datum	29.01.2025
Zeit	ca. 5 Stunden
Tätigkeiten	<p>Realisierung des Backends dokumentieren:</p> <ul style="list-style-type: none"> • Beschreibung der Backend-Architektur mit Spring Boot. • Dokumentation der Sicherheitsarchitektur mit JWT-basiertem Login. • Detaillierte Erklärung der API-Endpunkte für Benutzer, Zimmer und Buchungen. • Implementierung von Services, Repositories und Controller-Schichten dokumentieren. • Testszenarien für API-Aufrufe beschreiben (User-Login, Zimmerbuchung, Buchungsabruf).
Nächste Schritte	Realisierung des Frontends dokumentieren

Datum	30.01.2025
Zeit	ca. 6 Stunden
Tätigkeiten	<p>Realisierung des Frontends dokumentieren:</p> <ul style="list-style-type: none"> • Beschreibung der Projektstruktur mit React und Vite. • Erklärung der Routing-Architektur mit <i>Protected Routes</i> für Authentifizierung. • Dokumentation der wichtigsten Komponenten (<i>RoomSearch</i>, <i>RoomResult</i>, <i>RoomDetailsPage</i>). • Beschreibung der API-Kommunikation mit Axios zur Datenübertragung zwischen Frontend und Backend. • UI-Design-Aspekte und Nutzung von Tailwind für das Styling erläutern.
Nächste Schritte	Feinschliff der Dokumentation

Datum	31.01.2025
Zeit	ca. 8 Stunden
Tätigkeiten	<p>Feinschliff an der Dokumentation:</p> <ul style="list-style-type: none">• Vollständige Überprüfung und Korrektur aller Kapitel.• Ergänzungen und Verbesserungen in den Erklärungen zu technischen Konzepten.• Letzte Anpassungen am Sicherheitskonzept und der Backend-Architektur.• Finalisierung der Testfälle für Backend und Frontend.• Formatierung der Dokumentation für eine saubere und strukturierte Darstellung.
Nächste Schritte	Keine offenen Aufgaben mehr, Dokumentation ist abgeschlossen.