**Use Case Scenario: Movie Ticket Booking System**

**Entities**:

1. **Movie**
2. **Theater**
3. **Customer**
4. **Booking**

**Entities and Relationships:**

1. **Movie** (Many-to-Many with Theater)
   * Movie entity contains details about a movie.
2. **Theater** (Many-to-Many with Movie)
   * Theater entity contains details about a theater.
3. **Customer** (One-to-Many with Booking)
   * Customer entity contains details about customers.
4. **Booking** (Many-to-One with Movie, Many-to-One with Theater, Many-to-One with Customer)
   * Booking entity stores booking details and associates with movie, theater, and customer.

**Create the Entities**

**1. Movie Entity**

import javax.persistence.\*;

import java.util.List;

@Entity

public class Movie {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String genre;

@ManyToMany

@JoinTable(

name = "movie\_theater",

joinColumns = @JoinColumn(name = "movie\_id"),

inverseJoinColumns = @JoinColumn(name = "theater\_id"))

private List<Theater> theaters;

// Getters and Setters

}

**2. Theater Entity**

import javax.persistence.\*;

import java.util.List;

@Entity

public class Theater {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String location;

@ManyToMany(mappedBy = "theaters")

private List<Movie> movies;

// Getters and Setters

}

**3. Customer Entity**

import javax.persistence.\*;

import java.util.List;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@OneToMany(mappedBy = "customer")

private List<Booking> bookings;

// Getters and Setters

}

**4. Booking Entity**

import javax.persistence.\*;

@Entity

public class Booking {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String bookingDate;

private int seats;

@ManyToOne

@JoinColumn(name = "movie\_id")

private Movie movie;

@ManyToOne

@JoinColumn(name = "theater\_id")

private Theater theater;

@ManyToOne

@JoinColumn(name = "customer\_id")

private Customer customer;

// Getters and Setters

}

**Key REST Endpoints**

**1. Movie Endpoints:**

* **GET /api/movies**: Retrieve all movies.
* **GET /api/movies/{id}**: Retrieve details of a specific movie by its ID.
* **POST /api/movies**: Create a new movie.
* **PUT /api/movies/{id}**: Update a movie by ID.
* **DELETE /api/movies/{id}**: Delete a movie by ID.

**2. Theater Endpoints:**

* **GET /api/theaters**: Retrieve all theaters.
* **GET /api/theaters/{id}**: Retrieve details of a specific theater by its ID.
* **POST /api/theaters**: Create a new theater.
* **PUT /api/theaters/{id}**: Update a theater by ID.
* **DELETE /api/theaters/{id}**: Delete a theater by ID.

**3. Customer Endpoints:**

* **GET /api/customers**: Retrieve all customers.
* **GET /api/customers/{id}**: Retrieve details of a specific customer by their ID.
* **POST /api/customers**: Register a new customer.
* **PUT /api/customers/{id}**: Update a customer's information by ID.
* **DELETE /api/customers/{id}**: Delete a customer by ID.

**4. Booking Endpoints:**

* **GET /api/bookings**: Retrieve all bookings.
* **GET /api/bookings/{id}**: Retrieve details of a specific booking by its ID.
* **POST /api/bookings**: Create a new booking for a customer, specifying the movie and theater.
* **PUT /api/bookings/{id}**: Update booking details (e.g., change the number of seats or movie).
* **DELETE /api/bookings/{id}**: Delete a booking.

**Flow of the Application**

1. **Customer Registration**: A customer can register by providing their name and email.
2. **Movie and Theater Management**: Movies and theaters can be added, updated, or removed from the system. A movie can be associated with multiple theaters, and a theater can show multiple movies.
3. **Booking Process**: A customer can make a booking by specifying the movie, theater, and the number of seats. Each booking is linked to a specific movie, theater, and customer.
4. **Booking History**: A customer can view all their bookings, and booking details will include movie, theater, date, and number of seats.

**Summary of Relationships**

* **Movie** and **Theater** have a **Many-to-Many** relationship because a movie can be played in multiple theaters, and a theater can show multiple movies.
* **Customer** has a **One-to-Many** relationship with **Booking**, as one customer can make multiple bookings.
* **Booking** has a **Many-to-One** relationship with **Movie**, **Theater**, and **Customer**, as a booking refers to one movie, one theater, and one customer.

This system allows efficient movie ticket booking, with CRUD operations for movies, theaters, customers, and bookings, and provides the functionality to manage bookings, view booking history, and manage movie-theater assignments.

2. **Use Case Scenario: Hotel Booking System**

**Entities:**

1. **Hotel**
2. **Room**
3. **Booking**
4. **Customer**

**Relationships:**

1. A **Hotel** has many **Rooms** (One-to-Many).
2. A **Room** can have many **Bookings** (One-to-Many).
3. A **Booking** is made by a **Customer** (Many-to-One).

**1. Entity Classes (JPA)**

Let's define the entity classes and relationships using **Spring Data JPA**.

**Hotel.java (One-to-Many with Room)**

@Entity

public class Hotel {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String location;

@OneToMany(mappedBy = "hotel", cascade = CascadeType.ALL)

private List<Room> rooms;

// Getters and Setters

}

**Room.java (Many-to-One with Hotel and One-to-Many with Booking)**

@Entity

public class Room {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String roomNumber;

private String type; // e.g., Single, Double, Suite

private BigDecimal price;

@ManyToOne

@JoinColumn(name = "hotel\_id")

private Hotel hotel;

@OneToMany(mappedBy = "room")

private List<Booking> bookings;

// Getters and Setters

}

**Booking.java (Many-to-One with Customer and Room)**

@Entity

public class Booking {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private LocalDate checkInDate;

private LocalDate checkOutDate;

@ManyToOne

@JoinColumn(name = "room\_id")

private Room room;

@ManyToOne

@JoinColumn(name = "customer\_id")

private Customer customer;

// Getters and Setters

}

**Customer.java (One-to-Many with Booking)**

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String password; // This will be encoded

@OneToMany(mappedBy = "customer")

private List<Booking> bookings;

// Getters and Setters

}

**Summary of REST Endpoints:**

| **Entity** | **Endpoint** | **HTTP Method** | **Description** |
| --- | --- | --- | --- |
| Hotel | /api/hotels | GET | Get all hotels |
| Hotel | /api/hotels/{id} | GET | Get hotel by ID |
| Hotel | /api/hotels | POST | Create a new hotel |
| Hotel | /api/hotels/{id} | PUT | Update hotel |
| Hotel | /api/hotels/{id} | DELETE | Delete hotel |
| Room | /api/hotels/{hotelId}/rooms | GET | Get all rooms in a hotel |
| Room | /api/rooms/{id} | GET | Get room by ID |
| Room | /api/rooms | POST | Create a new room |
| Room | /api/rooms/{id} | PUT | Update room |
| Room | /api/rooms/{id} | DELETE | Delete room |
| Booking | /api/bookings | POST | Create a new booking |
| Booking | /api/bookings/{customerId} | GET | Get bookings by customer |
| Booking | /api/bookings/{id} | GET | Get booking by ID |
| Booking | /api/bookings/{id} | DELETE | Cancel booking |
| Customer | /api/customers/register | POST | Register a new customer |
| Customer | /api/customers/login | POST | Login customer and get JWT token |