Full java spring boot project hotel management system with code with full explanation for each module

**ChatGPT said:**

Sure! Below is an overview of a **Hotel Management System** built with **Spring Boot**. This project will include basic features such as room management, booking management, and user management. It will be explained with each module and how it can be implemented.

**Prerequisites**

1. **Java 11+** (for Spring Boot)
2. **Spring Boot Framework**
3. **Spring Data JPA** (for database interaction)
4. **Thymeleaf** (for the frontend, if you need HTML rendering)
5. **H2 Database** (or MySQL/PostgreSQL depending on your setup)

**Project Overview**

This project will have several modules:

1. **User Management**: Handles user registration, login, and roles.
2. **Room Management**: Handles room availability, pricing, and booking.
3. **Booking Management**: Handles booking, cancellation, and viewing of reservations.

**1. Set up Spring Boot Project**

You can create a Spring Boot project using **Spring Initializr** at [start.spring.io](https://start.spring.io/).

* **Dependencies**:
  + Spring Web
  + Spring Boot DevTools
  + Spring Data JPA
  + Thymeleaf (for web UI)
  + H2 Database (or MySQL/PostgreSQL)
  + Spring Security (for authentication and authorization)

Here’s the project structure that we’ll follow:

css

Copy

hotel-management-system

│

├── src/main/java/com/example/hotelmanagement

│ ├── controller/

│ ├── model/

│ ├── repository/

│ ├── service/

│ └── HotelManagementSystemApplication.java

│

├── src/main/resources/

│ ├── templates/

│ │ ├── index.html

│ │ ├── login.html

│ │ ├── room-list.html

│ │ └── booking.html

│ ├── application.properties

│

└── pom.xml

**2. pom.xml Dependencies (Maven)**

Here’s the required dependency configuration:

xml

Copy

<dependencies>

<!-- Spring Boot Starter Web (REST and UI) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot Starter for Thymeleaf -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<!-- Spring Boot Starter Data JPA (for database interaction) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database (in-memory for development) -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Security (for user management) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- Spring Boot Starter Test (testing) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

**3. Application Properties (application.properties)**

This file contains the configuration for the database (H2 for simplicity):

properties

Copy

# H2 Database configuration (for in-memory DB)

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

# Spring Security (default configurations)

spring.security.user.name=admin

spring.security.user.password=admin

**4. HotelManagementSystemApplication.java**

This is the entry point for your Spring Boot application.

java

Copy

package com.example.hotelmanagement;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HotelManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(HotelManagementSystemApplication.class, args);

}

}

**5. Model Classes**

Let’s start by creating the models representing **Room**, **Booking**, and **User**.

**5.1. Room Model (Room.java)**

java

Copy

package com.example.hotelmanagement.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

@Entity

public class Room {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String roomType;

private double price;

private boolean isAvailable;

// Getters and Setters

}

**5.2. Booking Model (Booking.java)**

java

Copy

package com.example.hotelmanagement.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import java.time.LocalDate;

@Entity

public class Booking {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long roomId;

private String customerName;

private LocalDate startDate;

private LocalDate endDate;

// Getters and Setters

}

**5.3. User Model (User.java)**

java

Copy

package com.example.hotelmanagement.model;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String role;

// Getters and Setters

}

**6. Repository Classes**

These are used to interact with the database via Spring Data JPA.

**6.1. Room Repository (RoomRepository.java)**

java

Copy

package com.example.hotelmanagement.repository;

import com.example.hotelmanagement.model.Room;

import org.springframework.data.jpa.repository.JpaRepository;

public interface RoomRepository extends JpaRepository<Room, Long> {

}

**6.2. Booking Repository (BookingRepository.java)**

java

Copy

package com.example.hotelmanagement.repository;

import com.example.hotelmanagement.model.Booking;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookingRepository extends JpaRepository<Booking, Long> {

}

**6.3. User Repository (UserRepository.java)**

java

Copy

package com.example.hotelmanagement.repository;

import com.example.hotelmanagement.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**7. Service Classes**

These classes contain the business logic for managing users, rooms, and bookings.

**7.1. Room Service (RoomService.java)**

java

Copy

package com.example.hotelmanagement.service;

import com.example.hotelmanagement.model.Room;

import com.example.hotelmanagement.repository.RoomRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class RoomService {

@Autowired

private RoomRepository roomRepository;

public List<Room> getAllRooms() {

return roomRepository.findAll();

}

public Room getRoomById(Long id) {

return roomRepository.findById(id).orElse(null);

}

public void addRoom(Room room) {

roomRepository.save(room);

}

public void updateRoom(Room room) {

roomRepository.save(room);

}

public void deleteRoom(Long id) {

roomRepository.deleteById(id);

}

}

**7.2. Booking Service (BookingService.java)**

java

Copy

package com.example.hotelmanagement.service;

import com.example.hotelmanagement.model.Booking;

import com.example.hotelmanagement.repository.BookingRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class BookingService {

@Autowired

private BookingRepository bookingRepository;

public void createBooking(Booking booking) {

bookingRepository.save(booking);

}

public void cancelBooking(Long bookingId) {

bookingRepository.deleteById(bookingId);

}

}

**7.3. User Service (UserService.java)**

java

Copy

package com.example.hotelmanagement.service;

import com.example.hotelmanagement.model.User;

import com.example.hotelmanagement.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public void registerUser(User user) {

userRepository.save(user);

}

}

**8. Controller Classes**

These classes handle HTTP requests and interact with the service layer.

**8.1. Room Controller (RoomController.java)**

java

Copy

package com.example.hotelmanagement.controller;

import com.example.hotelmanagement.model.Room;

import com.example.hotelmanagement.service.RoomService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class RoomController {

@Autowired

private RoomService roomService;

@GetMapping("/rooms")

public String getRooms(Model model) {

model.addAttribute("rooms", roomService.getAllRooms());

return "room-list";

}

}

**8.2. Booking Controller (BookingController.java)**

java

Copy

package com.example.hotelmanagement.controller;

import com.example.hotelmanagement.model.Booking;

import com.example.hotelmanagement.service.BookingService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.PostMapping;

@Controller

public class BookingController {

@Autowired

private BookingService bookingService;

@PostMapping("/bookRoom")

public String bookRoom(Booking booking) {

bookingService.createBooking(booking);

return "redirect:/rooms";

}

}

**8.3. User Controller (UserController.java)**

java

Copy

package com.example.hotelmanagement.controller;

import com.example.hotelmanagement.model.User;

import com.example.hotelmanagement.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.PostMapping;

@Controller

public class UserController {

@Autowired

private UserService userService;

@PostMapping("/register")

public String register(User user) {

userService.registerUser(user);

return "redirect:/login";

}

}

**9. Frontend (HTML Templates)**

* **room-list.html**: Displays available rooms
* **index.html**: Home page
* **login.html**: Login form
* **booking.html**: Booking page

These templates are created using **Thymeleaf** to render HTML dynamically based on user interactions.

**Conclusion**

This is a **basic hotel management system** built using **Spring Boot**, which includes room management, booking functionality, and user management (for registration). Each component (models, repositories, services, and controllers) is modularized for clarity and maintainability.

You can expand this project by adding more advanced features like:

* **Role-based access** (using Spring Security)
* **Payment integration**
* **Room availability check**
* **Email notifications**
* **Calendar views** for bookings

This system will serve as a good foundation for any more advanced hotel management system you might want to build.

**You said:**

give me the frontend module also i.e HTML template

**ChatGPT said:**

Sure! Below is the **frontend module** for the **Hotel Management System**, which includes several **HTML templates** using **Thymeleaf** (Spring Boot's default templating engine). These templates will render dynamic data on the frontend, making the web application interactive.

**1. index.html (Home Page)**

This page is the entry point of the application and might include a simple welcome message or links to the room listing and booking.

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hotel Management System</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h1 class="text-center">Welcome to the Hotel Management System</h1>

<div class="text-center">

<a href="/rooms" class="btn btn-primary">View Available Rooms</a>

<a href="/login" class="btn btn-secondary">Login</a>

</div>

</div>

</body>

</html>

**2. login.html (Login Page)**

This page allows users to log in. For simplicity, we’ll assume the login is handled with Spring Security.

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h2 class="text-center">Login</h2>

<form action="/login" method="post">

<div class="form-group">

<label for="username">Username</label>

<input type="text" class="form-control" id="username" name="username" required>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" class="form-control" id="password" name="password" required>

</div>

<button type="submit" class="btn btn-primary btn-block">Login</button>

</form>

</div>

</body>

</html>

**3. room-list.html (Room Listing Page)**

This page lists all available rooms. It will be populated with data from the backend (using Thymeleaf). Users can book a room by clicking the "Book" button.

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Available Rooms</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h2 class="text-center">Available Rooms</h2>

<table class="table table-bordered">

<thead>

<tr>

<th>Room Type</th>

<th>Price</th>

<th>Available</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<!-- Loop through rooms and display them dynamically -->

<tr th:each="room : ${rooms}">

<td th:text="${room.roomType}"></td>

<td th:text="${room.price}"></td>

<td th:text="${room.isAvailable ? 'Yes' : 'No'}"></td>

<td>

<a th:href="@{/bookRoom(id=${room.id})}" class="btn btn-success"

th:disabled="${!room.isAvailable}">Book Room</a>

</td>

</tr>

</tbody>

</table>

<a href="/" class="btn btn-secondary">Back to Home</a>

</div>

</body>

</html>

**4. booking.html (Booking Page)**

This page allows users to input booking details. It is shown after clicking the "Book Room" button from the room listing.

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Booking Room</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h2 class="text-center">Booking Information</h2>

<form action="/bookRoom" method="post">

<input type="hidden" name="roomId" th:value="${room.id}"/>

<div class="form-group">

<label for="customerName">Customer Name</label>

<input type="text" class="form-control" id="customerName" name="customerName" required>

</div>

<div class="form-group">

<label for="startDate">Start Date</label>

<input type="date" class="form-control" id="startDate" name="startDate" required>

</div>

<div class="form-group">

<label for="endDate">End Date</label>

<input type="date" class="form-control" id="endDate" name="endDate" required>

</div>

<button type="submit" class="btn btn-primary btn-block">Confirm Booking</button>

</form>

<a href="/rooms" class="btn btn-secondary mt-3">Back to Room List</a>

</div>

</body>

</html>

**5. register.html (User Registration Page)**

This page allows users to register for the system. It includes fields for the username, password, and a role (if you want to add roles like "Admin" or "Customer").

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Registration</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h2 class="text-center">Register</h2>

<form action="/register" method="post">

<div class="form-group">

<label for="username">Username</label>

<input type="text" class="form-control" id="username" name="username" required>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" class="form-control" id="password" name="password" required>

</div>

<div class="form-group">

<label for="role">Role</label>

<input type="text" class="form-control" id="role" name="role" required>

</div>

<button type="submit" class="btn btn-primary btn-block">Register</button>

</form>

<a href="/login" class="btn btn-secondary mt-3">Back to Login</a>

</div>

</body>

</html>

**6. 404.html (Error Page)**

This is an error page to handle cases where the requested URL does not exist.

html

Copy

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Page Not Found</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5 text-center">

<h2>Oops! Page Not Found (404)</h2>

<a href="/" class="btn btn-primary">Go Back to Home</a>

</div>

</body>

</html>

**7. Final Directory Structure for Templates**

The HTML templates should be stored in the src/main/resources/templates/ directory:

pgsql

Copy

hotel-management-system

├── src/main/resources

│ ├── templates

│ │ ├── index.html

│ │ ├── login.html

│ │ ├── room-list.html

│ │ ├── booking.html

│ │ ├── register.html

│ │ └── 404.html

**How Thymeleaf Works in Spring Boot**

1. **Model Population**: In each controller, we use the @GetMapping and @PostMapping annotations to map URL paths to specific methods. These methods populate the data model (Model) that gets passed to the HTML templates.
2. **Thymeleaf Syntax**: In the HTML files, Thymeleaf syntax (th:text, th:each, etc.) is used to dynamically inject data into the HTML. For example:
   * th:text="${room.roomType}" injects the room type data from the backend into the HTML.

**Conclusion**

This frontend setup gives you a basic hotel management system with pages for:

* Viewing available rooms
* Booking rooms
* User registration and login
* A home page with basic navigation

You can enhance this system by adding more features such as:

* Admin functionalities (room management, booking overview)
* Detailed booking history
* Payment integration

Let me know if you need any additional features or adjustments!

**You said:**

give me the table structure for the above code

**ChatGPT said:**

Sure! Below are the **table structures** for the database based on the code provided for the **Hotel Management System** using **Spring Boot**, **JPA**, and **Thymeleaf**.

We will have 3 main entities:

1. **Room**
2. **Booking**
3. **User**

Each entity will correspond to a table in the database. I'll provide the structure for each table based on the models provided earlier.

**1. Room Table**

The Room table stores details about each room in the hotel, such as its type, price, and availability.

sql

Copy

CREATE TABLE room (

id BIGINT AUTO\_INCREMENT PRIMARY KEY, -- Unique identifier for each room

room\_type VARCHAR(100) NOT NULL, -- Type of the room (e.g., Single, Double, Suite)

price DECIMAL(10, 2) NOT NULL, -- Price per night for the room

is\_available BOOLEAN NOT NULL -- Availability of the room (true or false)

);

* id: A unique identifier for each room (auto-incremented).
* room\_type: The type of the room (e.g., "Single", "Double", "Suite").
* price: The price of the room per night (a decimal value).
* is\_available: A boolean value indicating whether the room is available for booking.

**2. Booking Table**

The Booking table stores details about each booking made by a customer, including which room is booked, the customer’s name, and the booking dates.

sql

Copy

CREATE TABLE booking (

id BIGINT AUTO\_INCREMENT PRIMARY KEY, -- Unique identifier for each booking

room\_id BIGINT NOT NULL, -- Foreign key to the room being booked

customer\_name VARCHAR(255) NOT NULL, -- Name of the customer who made the booking

start\_date DATE NOT NULL, -- Start date of the booking

end\_date DATE NOT NULL, -- End date of the booking

FOREIGN KEY (room\_id) REFERENCES room(id) ON DELETE CASCADE

);

* id: A unique identifier for each booking (auto-incremented).
* room\_id: A foreign key that references the id of the Room table. It indicates which room was booked.
* customer\_name: The name of the customer who made the booking.
* start\_date: The start date of the booking.
* end\_date: The end date of the booking.
* The foreign key room\_id references the id of the room table. The ON DELETE CASCADE ensures that when a room is deleted, all associated bookings are also deleted.

**3. User Table**

The User table stores the details of users who are registered in the system. This table is essential for user authentication and registration.

sql

Copy

CREATE TABLE user (

id BIGINT AUTO\_INCREMENT PRIMARY KEY, -- Unique identifier for each user

username VARCHAR(255) NOT NULL, -- Username for login

password VARCHAR(255) NOT NULL, -- Password (hashed)

role VARCHAR(50) NOT NULL -- Role of the user (e.g., "ADMIN" or "USER")

);

* id: A unique identifier for each user (auto-incremented).
* username: The username used by the user to log in.
* password: The password associated with the username (this should ideally be stored in a hashed format for security).
* role: The role of the user (e.g., USER or ADMIN).

**4. SQL Schema for In-Memory Database (H2)**

If you're using **H2** as an in-memory database (as in the application.properties), the tables will be automatically created based on the JPA annotations, but you can also create them manually with the SQL provided above.

For example, if you want to run a script to set up your H2 database, you could include it in an init.sql file inside src/main/resources:

sql

Copy

-- src/main/resources/init.sql

-- Create Room table

CREATE TABLE room (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

room\_type VARCHAR(100) NOT NULL,

price DECIMAL(10, 2) NOT NULL,

is\_available BOOLEAN NOT NULL

);

-- Create Booking table

CREATE TABLE booking (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

room\_id BIGINT NOT NULL,

customer\_name VARCHAR(255) NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

FOREIGN KEY (room\_id) REFERENCES room(id) ON DELETE CASCADE

);

-- Create User table

CREATE TABLE user (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL,

password VARCHAR(255) NOT NULL,

role VARCHAR(50) NOT NULL

);

Make sure the init.sql file is located in the src/main/resources folder, so it will be executed when the application starts.

**5. Sample Data Insertion (Optional)**

You can also add some sample data to the tables in the init.sql file to populate the database when the application starts.

sql

Copy

-- Sample data for Room table

INSERT INTO room (room\_type, price, is\_available) VALUES

('Single', 100.00, TRUE),

('Double', 150.00, TRUE),

('Suite', 250.00, FALSE);

-- Sample data for User table (for testing purposes)

INSERT INTO user (username, password, role) VALUES

('admin', '{noop}adminpassword', 'ADMIN'),

('john\_doe', '{noop}userpassword', 'USER');

In the case above, '{noop}' is used to indicate that the password is not encoded (for simplicity). In a real application, you would encode the password using a hashing algorithm such as bcrypt.

**6. Foreign Key Constraints**

As shown in the Booking table, the room\_id column is a **foreign key** that references the id column in the Room table. This ensures referential integrity between the Booking and Room tables, meaning that a booking cannot exist without a valid room.

**Conclusion**

These are the basic **table structures** for your Hotel Management System:

* **Room Table**: Contains details about each room, including type, price, and availability.
* **Booking Table**: Stores booking information, linking to the Room table.
* **User Table**: Contains user authentication data, including usernames, passwords, and roles.

This structure can be expanded depending on the additional features you want to implement, such as adding payment tables, customer details, or historical booking data. You can modify the schema based on your exact requirements and database choice (H2, MySQL, PostgreSQL, etc.).

Let me know if you need any further help!