Le Mans Hotel Management System

Technical Documentation & Project Manual

Version: 1.0
Last Updated: December 2025
Document Type: Technical Specification & User Guide

Table of Contents

---

1. Introduction

1.1 Project Purpose

The **Le Mans Hotel Management System** is a modern, full-stack web application designed to revolutionize hotel operations. It provides a comprehensive platform for hotel administrators to manage rooms, bookings, and promotional offers while delivering an intuitive booking experience for guests.

1.2 Target Audience

- **Hotel Administrators**: Manage hotel operations, view analytics, and control bookings
- **Hotel Guests**: Browse rooms, make reservations, and manage their bookings
- **System Administrators**: Deploy and maintain the application infrastructure

1.3 Key Benefits

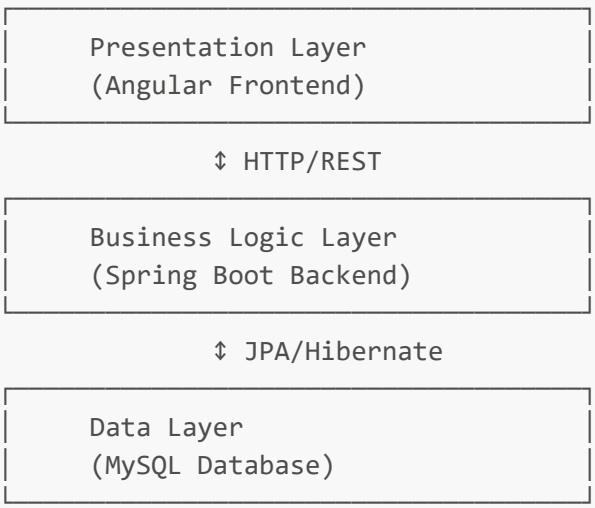
- ☑ **Streamlined Operations**: Centralized management of all hotel resources
- ☑ **Enhanced Guest Experience**: Intuitive booking interface with real-time availability
- ☑ **Security First**: JWT-based authentication with role-based access control
- ☑ **AI Integration**: Smart chatbot assistance for queries and support
- ☑ **Scalable Architecture**: Built on enterprise-grade Spring Boot framework

---

2. System Overview

2.1 Application Architecture

The system follows a **3-tier architecture**:



2.2 Core Modules

Module	Purpose	Access Level
Authentication	User login/registration, JWT token management	Public
Admin Dashboard	Analytics, revenue tracking, system overview	Admin Only
Room Management	CRUD operations for hotel rooms	Admin Only
Booking System	Reservation creation and management	User/Admin
Offer Management	Promotional campaigns and discounts	Admin Only
AI Chatbot	Natural language support assistance	User/Admin

3. Technology Stack

3.1 Backend Technologies

Component	Technology	Version
Framework	Spring Boot	3.2.5
Language	Java	17
Build Tool	Maven	3.x
Database	MySQL	8.0+
ORM	Hibernate (JPA)	6.4.x
Security	Spring Security + JWT	Latest

Component	Technology	Version
API Docs	SpringDoc OpenAPI	2.6.0
Email	Spring Mail	Latest

3.2 Frontend Technologies

Component	Technology	Version
Framework	Angular	17+
Language	TypeScript	5.9+
HTTP Client	Angular HttpClient	Latest
Routing	Angular Router	Latest
Forms	Angular Forms	Latest
Build Tool	Angular CLI	20.3+

3.3 Development Tools

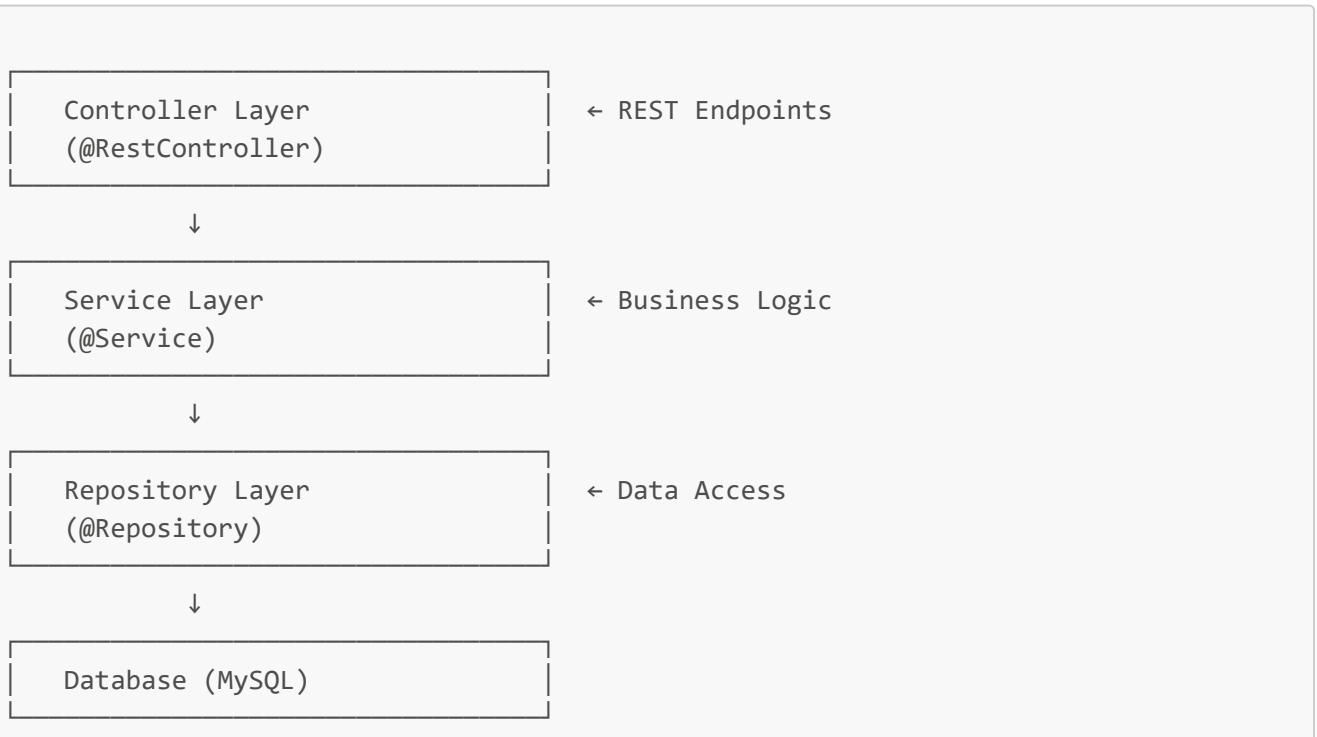
- **IDE Recommendations**: IntelliJ IDEA, VS Code
- **API Testing**: Swagger UI (built-in), Postman
- **Database Management**: MySQL Workbench

---

4. System Architecture

4.1 Backend Architecture

4.1.1 Layered Design



4.1.2 Security Architecture

- **Authentication**: JWT (JSON Web Token)
- **Session Management**: Stateless
- **Password Hashing**: BCrypt
- **Role-Based Access**: `ROLE_ADMIN`, `ROLE_USER`

4.1.3 Package Structure

```
com.example.le_mans_hotel/
├── ai/                    # AI chatbot controllers
├── configurations/        # App configuration
├── controller/            # REST controllers
├── dto/                   # Data Transfer Objects
├── exception/             # Exception handlers
├── model/                 # JPA entities
├── repository/            # Data repositories
├── scheduler/             # Scheduled tasks
├── security/              # Security config
└── service/               # Business logic
```

4.2 Frontend Architecture

4.2.1 Module Structure

```
src/app/
├── admin/                 # Admin module
│   ├── dashboard/
│   ├── add-room/
│   ├── manage-rooms/
│   ├── edit-room/
│   └── bookings/
├── booking-modal/         # Booking component
├── chatbot/               # AI chatbot
├── dining/                # Dining reservations
├── guards/                # Route guards
├── home/                  # Landing page
├── login/                 # Authentication
├── models/                # TypeScript interfaces
├── rooms/                 # Room browsing
├── services/              # HTTP services
└── shared/                # Shared components
```

---

5. Feature Specifications

5.1 Admin Features

5.1.1 Dashboard

- **Real-time Analytics**
  - Total revenue tracking
  - Booking statistics
  - Active guest count
  - Occupancy rates
- **Quick Actions**
  - Navigate to room management
  - Access offer creation
  - View all bookings

5.1.2 Room Management

- **Create Room**: Add new room with details and image upload
- **Edit Room**: Modify room information and pricing
- **Delete Room**: Remove rooms from inventory
- **View Rooms**: List all rooms with filtering options

5.1.3 Booking Management

- View all system bookings
- Filter by status (Pending, Confirmed, Cancelled)
- Approve or cancel reservations
- Track booking history

5.1.4 Offer Management

- Create promotional offers with discount percentages
- Set offer descriptions and titles
- View active offers
- Delete expired offers

5.2 User Features

5.2.1 Room Browsing

- View available rooms with high-quality images
- Filter by room type and price
- Check real-time availability
- View detailed room information

5.2.2 Booking System

- **Calendar-based Selection**: Choose check-in/check-out dates
- **Guest Count**: Specify number of persons (max 10)
- **Dining Options**: Add dining reservations (max 3 persons with dining)
- **Date Constraints**: Book from today onwards
- **Validation**: Real-time availability checks

5.2.3 Dining Reservations

- Browse fine dining options (French, Indian, Chinese, Japanese, Italian)
- Reserve tables for special occasions
- View cuisine details and descriptions

5.2.4 My Bookings

- View personal booking history
- Check booking status
- Download booking receipts (PDF)
- Cancel pending bookings

---

6. API Documentation

6.1 Authentication Endpoints

POST `/auth/register`

Description: Register a new user account

Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "securePassword123"
}
```

Response:

```
{
  "message": "User registered successfully",
  "userId": 1
}
```

POST `/auth/login`

Description: Authenticate user and receive JWT token

Request Body:

```
{
  "email": "john@example.com",
  "password": "securePassword123"
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "role": "USER",
  "expiresIn": 86400
}
```

6.2 Public Endpoints

GET `/public/rooms/{id}/image`

Description: Retrieve room image (No authentication required)

Response: Binary image data (JPEG/PNG)

6.3 Admin Endpoints (Requires ROLE_ADMIN)

GET `/admin/rooms`

Description: Retrieve all rooms

Response:

```
[
  {
    "id": 1,
    "roomType": "Deluxe Suite",
    "price": 250.00,
    "description": "Luxury suite with ocean view",
    "available": true
  }
]
```

POST `/admin/rooms`

Description: Create a new room

Request: Multipart form data

- `roomType`: String
- `price`: Double
- `description`: String
- `available`: Boolean
- `image`: File

PUT `/admin/rooms/{id}`

Description: Update room details

DELETE `/admin/rooms/{id}`

Description: Delete a room

GET `/admin/bookings`

Description: Retrieve all bookings

PUT `/admin/bookings/{id}/cancel`

Description: Cancel a booking

6.4 User Endpoints (Requires ROLE_USER)

GET `/user/rooms`

Description: Browse available rooms

POST `/user/bookings`

Description: Create a new booking

Request Body:

```
{
  "roomId": 1,
  "dishId": 2,
  "checkInDate": "2025-12-15",
  "checkOutDate": "2025-12-18",
  "noOfPerson": 2
}
```

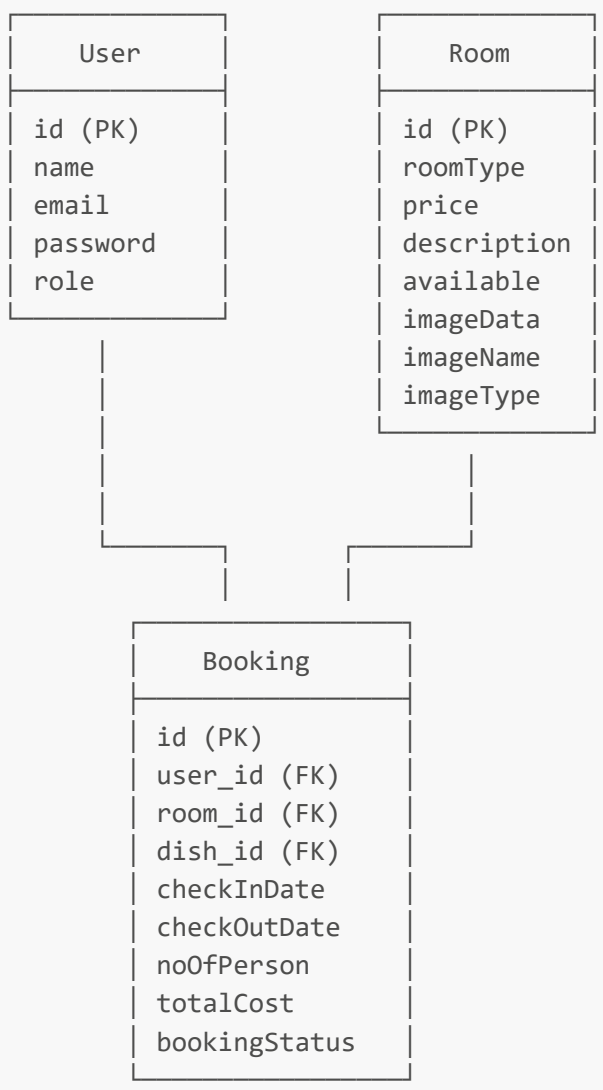
GET `/user/bookings`

Description: Retrieve user's booking history

---

7. Database Design

7.1 Entity Relationship Diagram



7.2 Table Schemas

Users Table

```
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role ENUM('USER', 'ADMIN') NOT NULL
);
```

Rooms Table

```
CREATE TABLE rooms (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    room_type VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    description TEXT,
    available BOOLEAN DEFAULT TRUE,
    image_data LONGBLOB,
    image_name VARCHAR(255),
    image_type VARCHAR(50)
);
```

Bookings Table

```
CREATE TABLE bookings (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    room_id BIGINT NOT NULL,
    dish_id BIGINT NOT NULL,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    no_of_person INT NOT NULL,
    total_cost DECIMAL(10,2) NOT NULL,
    booking_status ENUM('PENDING', 'CONFIRMED', 'CANCELLED'),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (room_id) REFERENCES rooms(id),
    FOREIGN KEY (dish_id) REFERENCES dishes(id)
);
```

---

8. Installation Guide

8.1 Prerequisites

Before installation, ensure you have:

- ☑ **Java Development Kit (JDK) 17 or higher**
  - Download from: https://www.oracle.com/java/technologies/downloads/
- ☑ **Node.js (v18+) and npm**
  - Download from: https://nodejs.org/
- ☑ **MySQL Server 8.0+**
  - Download from: https://dev.mysql.com/downloads/
- ☑ **Angular CLI**

```
npm install -g @angular/cli
```

8.2 Backend Setup

Step 1: Database Preparation

1. Start MySQL server
2. Open MySQL Workbench or terminal
3. Create the database:

```
CREATE DATABASE test_db;
```

Step 2: Configure Application

1. Navigate to `backend/src/main/resources/application.properties`
2. Update database credentials:

```
spring.datasource.url=jdbc:mysql://localhost:3306/test_db
spring.datasource.username=YOUR_USERNAME
spring.datasource.password=YOUR_PASSWORD
```

Step 3: Run Backend

```
cd backend
./mvnw spring-boot:run        # Mac/Linux
.\mvnw.cmd spring-boot:run    # Windows
```

The backend will start on `http://localhost:8080`

8.3 Frontend Setup

Step 1: Install Dependencies

```
cd frontend/lemans_hotel
npm install
```

Step 2: Start Development Server

```
ng serve
```

The frontend will start on `http://localhost:4200`

8.4 Verification

1. **Backend**: Visit http://localhost:8080/swagger-ui.html
2. **Frontend**: Visit http://localhost:4200
3. **Database**: Check MySQL Workbench for auto-created tables

---

9. Configuration

9.1 Environment Variables (Optional)

For production deployments, use environment variables:

```
# Database
export DB_URL="jdbc:mysql://localhost:3306/test_db"
export DB_USERNAME="your_username"
export DB_PASSWORD="your_password"

# JWT Security
export JWT_SECRET="your-256-bit-secret-key"

# Email (Optional)
export MAIL_USERNAME="your_email@gmail.com"
export MAIL_PASSWORD="your_app_password"
```

Refer to `ENV_SETUP.md` for detailed configuration options.

9.2 CORS Configuration

The backend is pre-configured to accept requests from `http://localhost:4200`.
For production, update `@CrossOrigin` annotations in controllers.

---

10. Development Workflow

10.1 Making Changes

Backend Development

1. Make code changes in `src/main/java`
2. Spring Boot DevTools will auto-reload
3. Test via Swagger UI or Postman

Frontend Development

1. Make changes in `src/app`

2. Angular CLI will auto-reload the browser
3. Test in browser at `http://localhost:4200`

10.2 Building for Production

Backend

```
./mvnw clean package
java -jar target/Hotel_Management-0.0.1-SNAPSHOT.jar
```

Frontend

```
ng build --configuration production
```

Output will be in `dist/` folder

Appendix

A. Default User Credentials

After first registration, manually update the role in the database:

```
UPDATE users SET role = 'ADMIN' WHERE email = 'your_email@example.com';
```

B. Troubleshooting

- **Port 8080 already in use**: Change `server.port` in `application.properties`
- **MySQL connection failed**: Verify MySQL service is running
- **JWT errors**: Ensure JWT secret is at least 256 bits (32 characters)

C. Support & Resources

- **Swagger API Docs**: http://localhost:8080/swagger-ui.html
- **Angular Docs**: https://angular.io/docs
- **Spring Boot Docs**: https://spring.io/projects/spring-boot

---

Document End

Le Mans Hotel Management System - Technical Documentation v1.0