# Barber Booking App – Documentation

The aim of this project is to develop a Spring Boot application for managing barbershop appointments. Customers can register, book a barber for a service, receive notifications, and leave reviews. Administrators or staff can manage barbers, services, and oversee bookings. The system uses **MySQL** for persistence, **Spring Data JPA** for database interactions, and **Swagger** for API documentation.

## 2. Business Requirements & MVP Features

### 2.1 Ten Business Requirements (Summary)

1. **User Registration and Login**
2. **Barber Management** (Admin/staff can add/update/delete barbers)
3. **Service Management** (Admin/staff can manage services like haircut, trim)
4. **Appointment (Booking) Creation** (Customers book barbers for specific times/services)
5. **Booking Management** (View, update, cancel appointments)
6. **Notification System** (Reminders, status updates)
7. **Payment Tracking** (Indicate if booking is paid, optional in MVP)
8. **Reviews & Ratings** (Customers review barbers/services)
9. **Reporting** (Daily/weekly appointment count, revenue)
10. **Scalability & Security** (Design to handle multiple locations, secure user data)

### 2.2 Five MVP Features

Out of the above ten, the **MVP** focuses on **five** key features:

1. **User Management**
    - Register new users (basic profile info).
    - Retrieve and manage existing users.
2. **Barber & Service Management**
    - Admin/staff can create, edit, delete barbers and services.
    - For services, set duration and price.
3. **Booking Creation & Scheduling**
    - Core logic: customers select a barber, service, and time to book an appointment.
4. **Booking Management (View, Update, Cancel)**
    - Customers can see and manage appointments.
    - Staff/barbers can view schedules.
5. **Basic Notification / Reminder**
    - Send or store notifications (e.g., appointment reminders).
    - This can be email-like or simply stored in the database for the MVP.

These five features represent the **minimum viable product** to cover the essential flow of user registration, scheduling, and basic communications.

## 3. System Architecture

### 3.1 Overview

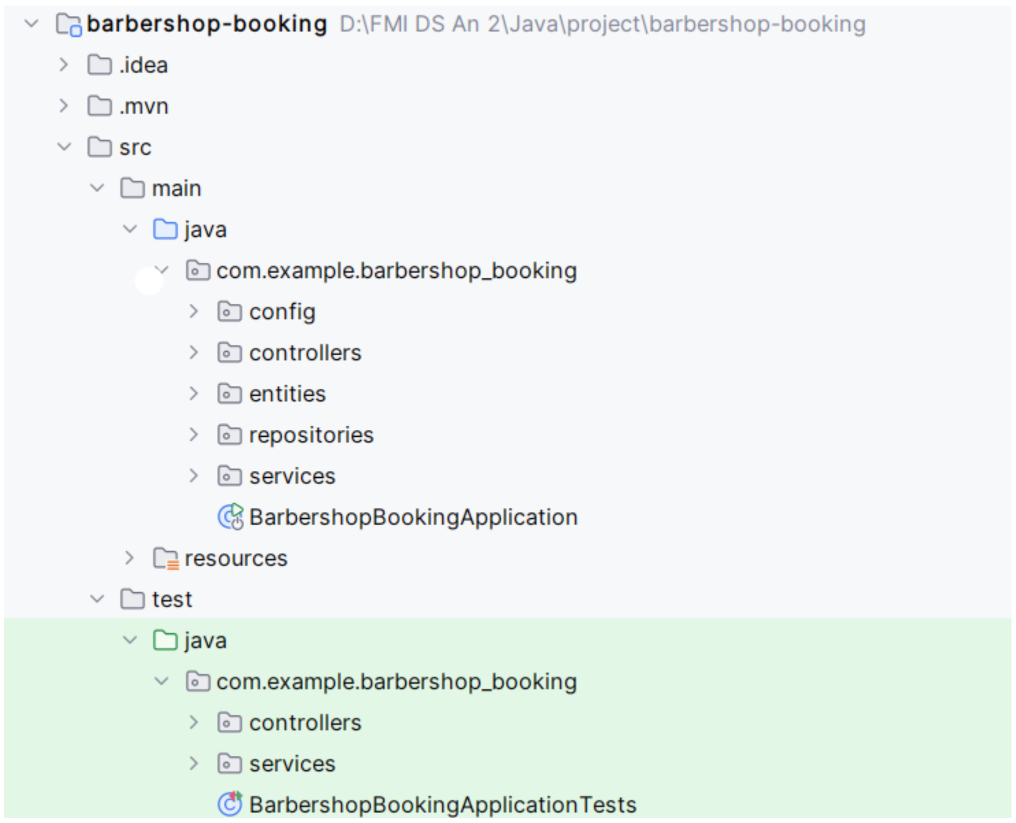
[ Client (Postman/Browser) ] ---> [ REST Controller Layer ] ---> [ Service Layer ] ---> [ Repository Layer ] ---> [ MySQL DB ]

- **Spring Boot** orchestrates the layers.
- **MySQL** stores users, barbers, services, bookings, notifications, reviews, etc.
- **Swagger** documents endpoints.

### 3.2 Main Technologies

- **Java 21+**
- **Spring Boot** (Web, Data JPA, Validation, DevTools)
- **MySQL**
- **Lombok**
- **Swagger / SpringDoc** (API docs)
- **JUnit + Mockito** (testing)

## 4. Project Structure

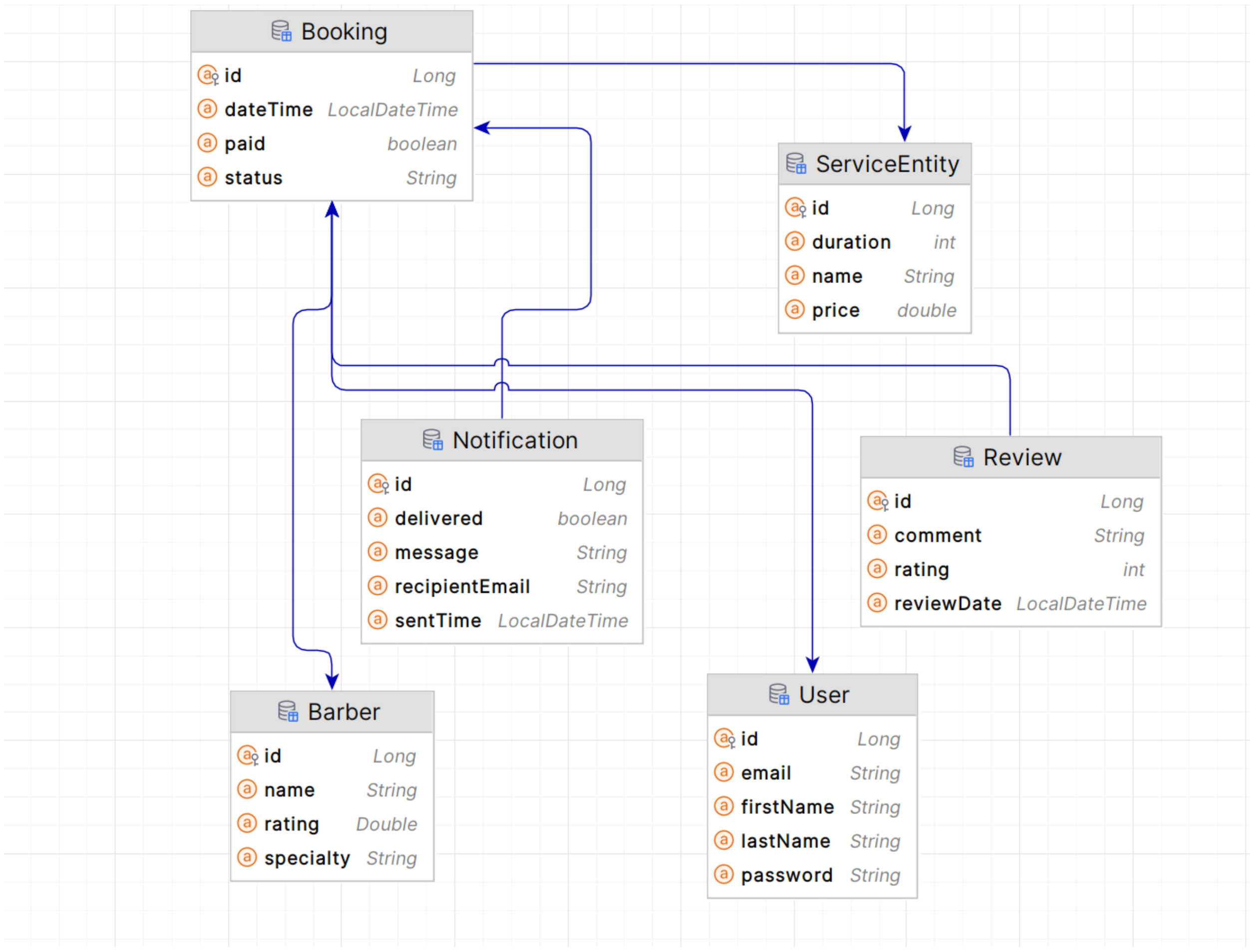


```
barbershop-booking D:\FMI DS An 2\Java\project\barbershop-booking
  .idea
  .mvn
  src
    main
      java
        com.example.barbershop_booking
          config
          controllers
          entities
          repositories
          services
          BarbershopBookingApplication
      resources
    test
      java
        com.example.barbershop_booking
          controllers
          services
          BarbershopBookingApplicationTests
```

## 5. Database Model

There are **six entities**:

1. **User** – one-to-many with **Booking**
2. **Barber** – one-to-many with **Booking**
3. **ServiceEntity** – many-to-many with **Booking**
4. **Booking** – many-to-one with **User**, many-to-one with **Barber**, many-to-many with **ServiceEntity**
5. **Notification** – many-to-one with **Booking**
6. **Review** – many-to-one with **Booking**



## 6. Installation & Setup

1. **Clone** the repository or download the source.
2. **Create** a MySQL database, e.g. barbershopdb.
3. **Configure** application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/barbershopdb?useSSL=false&serverTimezone=UTC
spring.datasource.username=YOUR_USER
spring.datasource.password=YOUR_PASSWORD
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

4. **Build**: `mvn clean install`
5. **Run**: `mvn spring-boot:run`
6. Access the app at **http://localhost:8080**.

## 7. Endpoints (Implementing the MVP Features)

Below is a sumamry of the endpoint reference. (See the full list in the project's Swagger UI.)

### 7.1 User

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/users | Create a new user (registration) |
| GET | /api/users | Get all users |
| GET | /api/users/{id} | Get user by ID |
| PUT | /api/users/{id} | Update user |
| DELETE | /api/users/{id} | Delete user |
| GET | /api/users/byName | Get users by name (containing) |

### 7.2 Barber & Service

- **Barber** Endpoints (/api/barbers):
  - **POST**: create barber
  - **GET**: get all
  - **GET**: get by ID
  - **PUT**: update
  - **DELETE**: remove
  - Additional queries: by name, by rating.
- **Service** Endpoints (/api/services):
  - **POST**: create new service
  - **GET**: list all services
  - **GET**: by ID

    - **PUT**: update
    - **DELETE**: remove
    - Additional queries: by price, by duration.

### 7.3 Booking

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/bookings | Create a new booking (with userId, barberId, serviceIds, date) |
| GET | /api/bookings | Get all bookings |
| GET | /api/bookings/{id} | Get booking by ID |
| PUT | /api/bookings/{id} | Update booking (reschedule) |
| DELETE | /api/bookings/{id}/cancel | Cancel booking |

### 7.4 Booking Management (View, Update, Cancel)

*(Handled by the same booking endpoints above: GET to view, PUT to update, DELETE to cancel.)*

### 7.5 Notification (Basic Notification/Reminder)

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/notifications | Create/store a notification |
| GET | /api/notifications | Get all notifications |
| POST | /api/notifications/sendReminder/{id} | Send (simulate) a reminder for a booking ID |
| GET | /api/notifications/byDelivered | Filter notifications by delivered boolean |

**7.6. Reviews**: GET(id), PUT(id), DELETE(id) POST(id), GET (all), GET(byRating), GET (byComment)

## 9. API Documentation with Swagger

Once running:

- **Swagger UI**: http://localhost:8080/swagger-ui.html
- **OpenAPI**: http://localhost:8080/api-docs

Or open the HTML file *api-docs.html*.

# 10. Testing

## 10.1 Unit Tests (Services)

- Each **service** has a **Mockito**-based test mocking the repository layer (e.g., UserServiceTest, BarberServiceTest, etc.).
- Validate business logic in **isolation**.

## 10.2 Controller Tests

- @WebMvcTest(ControllerClass) with **MockMvc** to check REST endpoints.
- Ensures the correct status codes, JSON responses, or error messages.

To run tests, use *mvn clean test.* All tests pass.

The database can be popullated with pre-defined data using file *Queries.SQL.*