Downloaded Spring Project template from start.spring.io  
Spring web, spring data JPA, MySql driver, JDBC API, Lombok

Opened in Eclipse

In MySQL:

create database hotel;

DB will be created.(No table created yet, )

In Eclipse:

application.properties

Added driver, MySQL endpoint with created DB name, username, password, port.

Create few packages.

Constants, Dao, JWT, POJO(model), Rest, RestImpl, Service, ServiceImpl, Utils, Wrapper

Let’s trace the journey of a new user through your Spring Boot application, step by step. Think of it like a relay race, where each layer passes the baton forward until the registration is complete.

### ⚙️ ****Config & Security Layer****

**Where:** com.hotel.config **Includes:** SecurityConfig.java **Purpose:**

* Houses configurations like Spring Security setup, CORS policies, etc.

### 🧰 ****Utility Layer****

**Where:** com.hotel.utils **Includes:** HotelUtils.java **Purpose:**

* Provides helper methods that can be reused across different layers.

### 📜 ****Constants & Wrapper Layers****

**Where:** com.hotel.constants, com.hotel.wrapper **Purpose:**

* Constants: Stores static values (e.g., messages, error codes).
* Wrapper: Often used for custom response objects or data transfer wrappers (this one’s empty for now!).

### 🔨 0****. Application Entry Point****

**Where:** com.hotel **Includes:** HotelManagementApplication.java **Purpose:**

* Main class that launches the Spring Boot application with @SpringBootApplication.

### 🏁 1. ****Controller Layer (REST Entry Point)****

**Where:** com.hotel.rest and com.hotel.restImpl

**Includes:** UserRest.java, UserRestImpl.java **Purpose:**

* Exposes APIs to clients (e.g., frontend, Postman, mobile apps).
* Accepts HTTP requests and returns HTTP responses.
* **Communicates with the Service layer to get data.**

**File:** UserRestImpl.java

* The frontend or client sends a POST request (e.g., /user/signup) with user data in JSON format.
* The controller receives this HTTP request and invokes the appropriate method in the **Service layer**.

### 🧠 2. ****Service Layer (Business Logic)****

**Where:** com.hotel.service and com.hotel.serviceImpl

**Includes:** UserService.java, UserServiceImpl.java

**Purpose:**

* Contains business logic.
* Implements the services used by the REST layer.
* **Bridges the controller and DAO layers.**

**File:** UserServiceImpl.java

* Validates incoming data (e.g., are fields null, is the email valid).
* May check whether a user with the same email already exists by calling a DAO method.
* Encrypts the password before saving.
* Constructs a User object and passes it to the DAO layer for persistence.

### 💾 3. ****DAO Layer (Database Access)****

**Where:** com.hotel.dao **Includes:** UserDao.java **Purpose:**

* Provides interfaces for CRUD operations with the database.
* **Extends Spring Data JPA repositories to interact with your** User **entity.**

**File:** UserDao.java

* This is typically an interface extending JpaRepository<User, Integer>.
* It handles the actual interaction with the database.
* save(user) converts your User object into an INSERT query and stores it in the database.

### 🧱 4. ****Model/Entity Layer (a.k.a. Entity Class or POJO Layer)****

**Where:** com.hotel.POJO **Includes:** User.java **Purpose:**

* Represents the database structure.
* Each class is mapped to a table using JPA annotations.
* **Think of these as the “shapes” of your data.**

**File:** User.java

* This is the structure used to persist and retrieve data from the database.
* Spring JPA maps the object fields to database columns and vice versa.

### 📨 5. ****Response Back to Client****

* Once stored, the service returns a response (Success or User already exists, etc.) back to the controller, which wraps it in an HTTP response and sends it back to the client.

🔁 Flow Summary:

Client ➝ Controller ➝ Service ➝ DAO ➝ Entity ➝ Database

⬅︎ ⬅︎ ⬅︎ ⬅︎

Response (e.g., Success message)