**User CRUD Application - Documentation**

**Introduction**

This is a **Spring Boot-based User Management System** that provides CRUD operations (Create, Read, Update, Delete) for managing users. It leverages **Spring Data JPA** for database interactions and follows the RESTful API architecture.

**Technologies Used**

* **Spring Boot** - For building the application.
* **Spring Data JPA** - For database interaction.
* **MySQL** - As the relational database.
* **Postman** - For API testing.
* **SQL Workbench** - For managing the database.
* **Eclipse IDE** - For coding and development.

**Project Structure**

**1. Package Structure**

org.user.user\_crud

│── controller # Contains REST API controllers

│── dao # Handles database interaction logic

│── dto # Data Transfer Object (Entity class)

│── repository # Repository interface for JPA

│── UserCrudApplication.java # Main Spring Boot application

**2. Entity Class - User**

Located in org.user.user\_crud.dto.User.java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int userId;

private String userName;

private String email;

private String pwd;

private long phNo;

// Getters and Setters

}

**Database Configuration**

**1. Create MySQL Database**

CREATE DATABASE user\_crud\_db;

**2. Configure application.properties**

spring.datasource.url=jdbc:mysql://localhost:3306/user\_crud\_db

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**API Endpoints**

**1. Create User**

**Endpoint:** POST /users/save

* **Request Body:**

{

"userName": "John Doe",

"email": "john.doe@example.com",

"pwd": "password123",

"phNo": 9876543210

}

* **Response:**

User saved

**2. Retrieve User by ID**

**Endpoint:** GET /users/find/{id}

* **Response Example:**

{

"userId": 1,

"userName": "John Doe",

"email": "john.doe@example.com",

"pwd": "password123",

"phNo": 9876543210

}

**3. Retrieve All Users**

**Endpoint:** GET /users/findAll

* **Response Example:**

[

{

"userId": 1,

"userName": "John Doe",

"email": "john.doe@example.com",

"pwd": "password123",

"phNo": 9876543210

},

{

"userId": 2,

"userName": "Alice",

"email": "alice@example.com",

"pwd": "alice123",

"phNo": 9988776655

}

]

**4. Update User**

**Endpoint:** PUT /users/update/{id}

* **Request Body:**

{

"userName": "Updated Name"

}

* **Response:**

User with ID {id} updated successfully.

**5. Delete User by ID**

**Endpoint:** DELETE /users/delete/{id}

* **Response:**

User with ID {id} deleted successfully.

**6. Delete All Users**

**Endpoint:** DELETE /users/deleteAll

* **Response:**

All users deleted successfully.

**7. Count Users**

**Endpoint:** GET /users/count

* **Response:**

Total number of users: 5

**Implementation Details**

**1. Repository Layer**

Located in org.user.user\_crud.repository.UserRepository.java

public interface UserRepository extends JpaRepository<User, Integer> {

}

**2. Data Access Layer (DAO)**

Located in org.user.user\_crud.dao.UserDao.java

public void updateUser(int id, User user) {

User existingUser = repo.findById(id)

.orElseThrow(() -> new RuntimeException("User with ID " + id + " not found"));

if (user.getUserName() != null) {

existingUser.setUserName(user.getUserName());

}

if (user.getEmail() != null) {

existingUser.setEmail(user.getEmail());

}

if (user.getPwd() != null) {

existingUser.setPwd(user.getPwd());

}

if (user.getPhNo() != 0) {

existingUser.setPhNo(user.getPhNo());

}

repo.save(existingUser);

}

**3. Controller Layer**

Located in org.user.user\_crud.controller.UserController.java

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserDao userDao;

@PostMapping("/save")

public ResponseEntity<String> saveUser(@RequestBody User user) {

userDao.saveUser(user);

return ResponseEntity.ok("User saved");

}

}

**Running the Application**

**1. Steps to Run**

1. Start MySQL server and create the database.
2. Configure application.properties with your database credentials.
3. Open the project in Eclipse IDE.
4. Run UserCrudApplication.java as a **Spring Boot Application**.
5. Use **Postman** to test the endpoints.

**2. Testing with Postman**

* Open Postman and set **Content-Type: application/json** for POST/PUT requests.
* Use **localhost:8080/users** as the base URL for API calls.

**Conclusion**

This User CRUD application provides a robust and flexible API for managing users in a MySQL database. The use of **Spring Boot, Spring Data JPA, and RESTful APIs** ensures efficiency and scalability. Further enhancements like **JWT Authentication, Role-based Access Control (RBAC), and frontend integration** can be added in the future.

**Future Improvements**

✅ Add validation for user inputs. ✅ Implement JWT-based authentication. ✅ Improve error handling with custom exception handlers. ✅ Create a frontend using React or Angular.