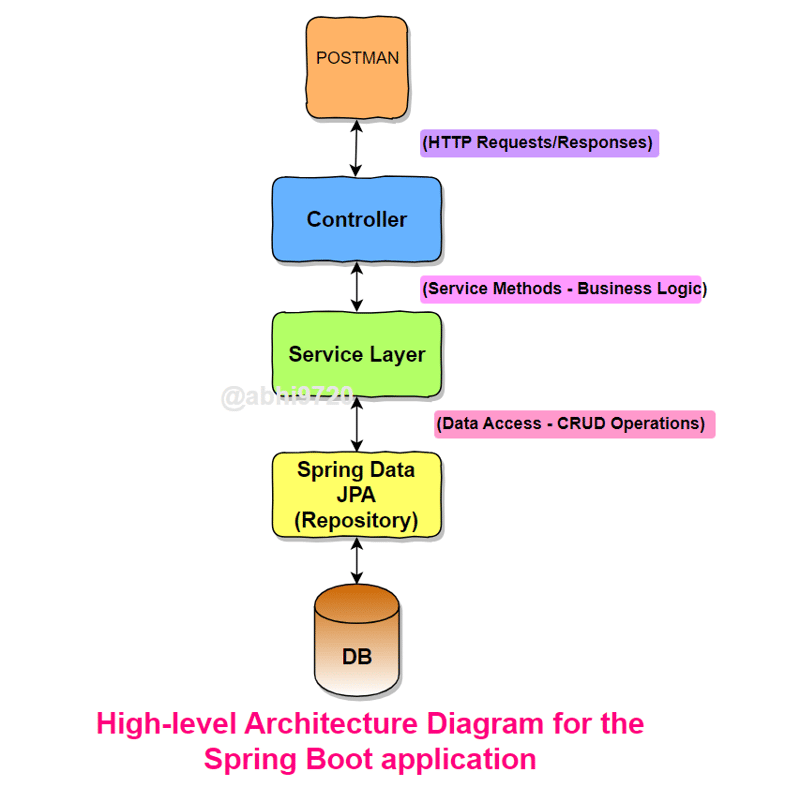
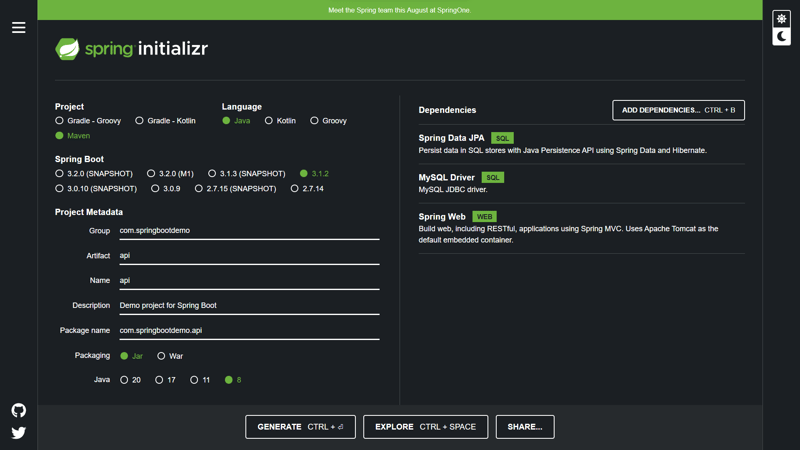
**Spring Boot Project High-level Architecture Diagram:**

The high-level architecture diagram depicts the flow of data and interactions within the Spring Boot application. It illustrates how frontend UI/Postman communicates with the Spring Boot API, which further coordinates with the service layer and Spring Data JPA to perform CRUD operations on the MySQL database.

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Frmx3u7tvnlvywajkuhea.png)

**Step 1: Setting up the Project**

Let's begin by creating a new Spring Boot project:  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fuser-images.githubusercontent.com%2F68281476%2F257251043-2685ca7c-c45a-4df5-ab3a-210ec400f0e0.png)

1. Go to Spring Initializr (<https://start.spring.io/>) to create a new Spring Boot project.
2. Add the required dependencies:
   * Spring Web: Provides the necessary components for building web applications.
   * Spring Data JPA: Simplifies database access using JPA (Java Persistence API).
   * MySQL Driver: Allows Spring Boot to communicate with the MySQL database.
3. Generate the project and import it into your IDE.

**Step 2: Database Configuration**

Now, let's configure the database to store our data. In this example, we will use MySQL:

1. Install MySQL on your local machine if you haven't already. You can download it from the official website: <https://dev.mysql.com/downloads/installer/>.
2. Create a new database nec.

* Create database nec;
* Use nec;

**Step 3: Creating the Entity**

In Spring Boot, an entity represents a table in the database. Let's create an entity class representing our "users" table:

// User.java

import javax.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

private String password;

private String name;

@Id

private String email;

private String address;

// Getters and setters

}

**Step 4: Defining Application Properties**

The src/main/resources/application.properties

# Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/nec

spring.datasource.username=root

spring.datasource.password=rajesh

# Hibernate Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

server.port=10000

**Step 5: Creating the Repository**

Src/main/java/com/banking/dao/UserRepository.java

// UserRepository.java

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

@Repository

public interface UserRepository extends JpaRepository<Employee, Long> {

}

**Step 6: Implementing the Service Layer**

The service layer contains the business logic and coordinates with the repository to perform database operations. Let's create the service class for our API:

// UserService.java

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User createUser(User user) {

return userRepository.save(user);

}

public List<User> getAllUsers() {

return userRepository.findAll();

}

public User getUserById(String id) {

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

}

public User updateUser(String id, Employee user) {

user.setId(id);

return userRepository.save(user);

}

public void deleteUser(String id) {

userRepository.deleteById(id);

}

}

**Step 7: Creating the Controller**

The controller handles HTTP requests and invokes the service methods. It remains unchanged from the previous blog:

// UserController.java

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/v1/employees")

public class UserController {

@Autowired

private UserService userService;

@PostMapping

public User createUser(@RequestBody User user) {

return userService.createUser(user);

}

@GetMapping

public List<User> getAllUsers() {

return userService.getAllUsers();

}

@GetMapping("/{id}")

public User getUserById(@PathVariable Long id) {

return userService.getUserById(id);

}

@PutMapping("/{id}")

public User updateUser(@PathVariable Long id, @RequestBody User user) {

return userService.updateUser(id, user);

}

@DeleteMapping("/{id}")

public void deleteUser(@PathVariable Long id) {

userService.deleteUser(id);

}

}

**Step 8: Running the Application**

1. Right click on main file and run as java aaplication

**Step 9: Testing the API Endpoints**

Testing using postman