Spring Framework provides a module for its data access layer Spring Data JPA, Spring Data JPA(Java persistence API) makes it easier to work with relational databases.

In this article, we will learn to create a Restful CRUD service in Spring Boot to perform CRUD operations on a database entity.

**This is the step by step guide which will cover:**

1. Spring Boot JPA Benefits
2. Create Spring Boot Application
3. Dependencies
4. Connection to Database
5. Creating Entity & Repository
6. Creating Service & Controller (Api Endpoints)
7. Testing our CRUP api’s

**Benefits of using JPA repositories :**

**Increases Productivity**: JPA reduces low-level database implementations, It makes developers focus more on business logic by providing auto-configuration.

**Reduces Boilerplate Code**: With JPA you write less boilerplate code for tasks like database connection management, transaction handling, entity-to-table mapping

**Portability**: JPA is a standard specification, your code will be compatible with all database vendors like H2, Postgres, MySQL etc.

**Transaction Management:**Both SpringBoot and JPA make it easier to do transaction management by providing auto configurations and annotations.

Now, let's move on to the practical example which will demonstrate the implementation of CRUD operation using Spring Boot Data JPA.

**Create Spring Boot Application**

If you want help in creating the springBoot application — check out this article which explains how to get started with the springboot application from scratch.

**Dependencies :**

We need to add the following dependencies in our pom.xml to use spring boot data JPA and h2 database.

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>

**Database Connection :**

# H2 in-memory database configuration for tests

spring.datasource.url=jdbc:h2:mem:testdb;DB\_CLOSE\_DELAY=-1;DB\_CLOSE\_ON\_EXIT=FALSE

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

**Creating JPA Entity :**

This is the very first step, We are going to create one simple Entity class UserEntity.

package com.exampleJpa;  
  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
  
@Entity  
@Table(name = "user\_details")  
public class UserEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(name = "user\_name")  
 private String userName;  
  
 @Column(name = "user\_address")  
 private String userAddress;  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public String getUserAddress() {  
 return userAddress;  
 }  
  
 public void setUserAddress(String userAddress) {  
 this.userAddress = userAddress;  
 }  
}

We have used a few annotations here to create our entity class, let's learn about them.

**Entity**: This annotation specifies that the class is an entity and that the class can be mapped to the database table.

**Table**: This annotation allows you to specify details about the table, as in this example we have mentioned the table name, using parameter name along with table annotation.

**Column**: This annotation allows to specify properties of the database column.

**Id**: Indicates that the member field below is the primary key of the entity.

Let's check our H2 database (the table should be created in its user\_details).

***Note: After adding Entity Class we just need to run the application, it will create a table user\_entity in our database.***

Now our user table is created we will perform CRUD operation on this user table using the JPA repository

**Creating User Repository :**

package com.exampleJpa;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
}

In the above code, we have generated a repository for our Entity class UserEntity using SpringBoot’s JpaRepository.

Now let’s add a controller and Service classes to implement the CRUD operation on your UserEntity class.

**Creating Service Interface**

package com.exampleJpa.service;  
  
import com.exampleJpa.UserEntity;  
  
import java.util.List;  
  
  
public interface JPAExampleService {  
 UserEntity saveUserData(UserEntity userEntity);  
 List<UserEntity> getUserData();  
 UserEntity updateUserData(UserEntity userEntity);  
 void deleteUserData(Long id);  
}

**Creating Service Implementation**

package com.exampleJpa.service;  
  
import com.exampleJpa.UserEntity;  
import com.exampleJpa.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatusCode;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class JPAExampleServiceImpl implements JPAExampleService {  
  
 @Autowired  
 UserRepository userRepository;  
  
 @Override  
 public UserEntity saveUserData(UserEntity userEntity) {  
 return userRepository.save(userEntity);  
 }  
  
 @Override  
 public List<UserEntity> getUserData() {  
 return userRepository.findAll();  
 }  
  
 @Override  
 public UserEntity updateUserData(UserEntity userEntity) {  
 return userRepository.save(userEntity);  
 }  
  
 @Override  
 public void deleteUserData(Long id) {  
 userRepository.deleteById(id);  
 }  
}

We have not written any DB queries in our code to add, update, get, or delete data from the database, JPARepository provides ready-to-use methods for CRUD operation which we just need to make calls.

**Creating Controller class**

package com.exampleJpa;  
  
import com.exampleJpa.service.JPAExampleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatusCode;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
public class JPAExampleController {  
 @Autowired  
 JPAExampleService jpaExampleService;  
  
  
 @PostMapping(value = "/user")  
 public ResponseEntity saveUserData(@RequestBody UserEntity userEntity){  
 try {  
 jpaExampleService.saveUserData(userEntity);  
 return new ResponseEntity<>(HttpStatusCode.valueOf(200));  
 }  
 catch (Exception e){  
 return new ResponseEntity<>(HttpStatusCode.valueOf(500));  
 }  
 }  
  
 @GetMapping(value = "/users")  
 public ResponseEntity<List<UserEntity>> getUserData(){  
 try {  
 return new ResponseEntity<>(jpaExampleService.getUserData(),HttpStatusCode.valueOf(200));  
 }  
 catch (Exception e){  
 return new ResponseEntity<>(HttpStatusCode.valueOf(500));  
 }  
 }  
  
 @PutMapping(value = "/user")  
 public ResponseEntity updateUserData(@RequestBody UserEntity userEntity){  
 try {  
 jpaExampleService.updateUserData(userEntity);  
 return new ResponseEntity<>(HttpStatusCode.valueOf(200));  
 }  
 catch (Exception e){  
 return new ResponseEntity<>(HttpStatusCode.valueOf(500));  
 }  
 }  
  
 @DeleteMapping(value = "/user")  
 public ResponseEntity updateUserData(@RequestParam(value = "userId") Long userId){  
 try {  
 jpaExampleService.deleteUserData(userId);  
 return new ResponseEntity<>(HttpStatusCode.valueOf(200));  
 }  
 catch (Exception e){  
 return new ResponseEntity<>(HttpStatusCode.valueOf(500));  
 }  
 }  
}I

In the above class we have created APIs for CRUD operation on UserEntity

This was a simple example with only CRUD operation, which is a great way of starting to learn Spring Data JPA.

There is a lot to learn about spring data JPA,