Spring Data JPA

- JPA – Java Persistence API; it is a standard from “Oracle”
- Used to store objects in DB & convert DB rows into objects
- It deals with object since it incorporates Object-oriented applications

Provider to implement JPA:

(I) Hibernate
(II) EclipseLink
(III) Open JPA

JPA standard provides two key interfaces to perform CRUD operations:

(i) EntityManagerFactory
(ii) EntityManager

It is implemented as:

- Objects are created that will use CRUD methods
- Internally the provider does the SQL operation
- Data class will be created to reflect the table

Pre-requisite:

1. DB & respective tables should be created
2. Ensure table has all fields
3. If needed keep prior record

Example:

Plan 1: Connect with DB thru Spring Application

Create a Maven Project thru Spring initializr with needed dependencies & import the project in Eclipse

Pom.xml

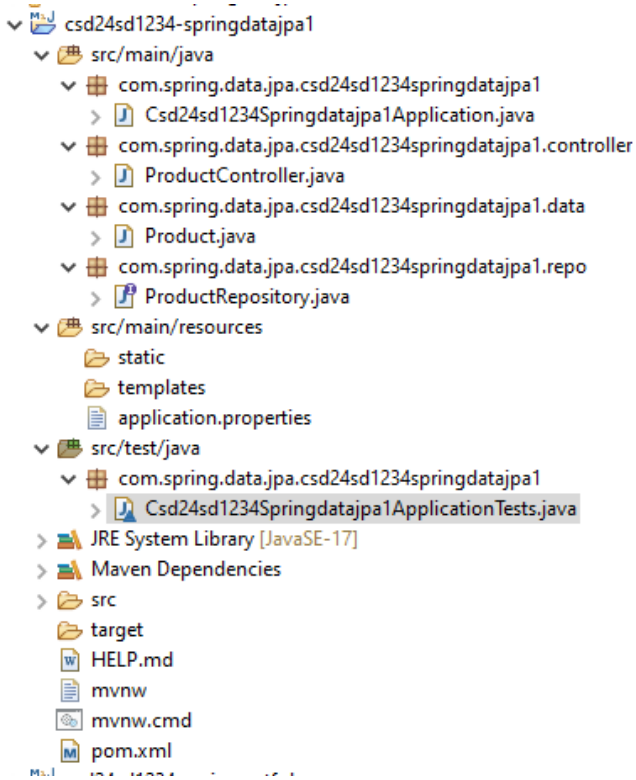
```
<?xml version="1.0" encoding="UTF-8"?>
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>

    <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.spring.data.jpa</groupId>
    <artifactId>csd24sd1234-springdatajpa1</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>csd24sd1234-springdatajpa1</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>

    <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
```

```
        <dependency>

    <groupId>org.springframework.boot</groupI
d>
            <artifactId>spring-boot-starter-
web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-
j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>

    <groupId>org.springframework.boot</groupI
d>
            <artifactId>spring-boot-starter-
test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>

    <groupId>org.springframework.boot</groupI
d>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
```

```
        </plugins>
    </build>

</project>
```

Folder Structure:



Main Application:

```
package com.spring.data.jpa.csd24sd1234springdatajpa1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
```

```
public class
Csd24sd1234Springdatajpa1Application {

    public static void main(String[] args) {

    SpringApplication.run(Csd24sd1234Springda
tajpa1Application.class, args);
    }

}
```

Product.java

```
package
com.spring.data.jpa.csd24sd1234springdatajpa1
.data;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class Product {

    @Id
    private int productid;
    private String productname;
    private double price;

    public int getProductid() {
        return productid;
    }
    public void setProductid(int productid) {
        this.productid = productid;
    }
```

```
    public String getProductname() {
        return productname;
    }
    public void setProductname(String
productname) {
        this.productname = productname;
    }

    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Product [productid=" +
productid + ", productname=" + productname +
", price=" + price + "]";
    }

}
```

ProductRepository Interface

```
package
com.spring.data.jpa.csd24sd1234springdatajpa1
.repo;
```

```
import
org.springframework.data.repository.CrudRepos
itory;

import
com.spring.data.jpa.csd24sd1234springdatajpa1
.data.Product;

public interface ProductRepository extends
CrudRepository<Product, Integer>{

}
```

Application.Properties

```
spring.application.name=csd24sd1234-
springdatajpa1
spring.datasource.name=mydb
spring.datasource.url=jdbc:mysql://localhost:
3306/sampledb
spring.datasource.username=root
spring.datasource.password=root1
```

Test file:

```
package
com.spring.data.jpa.csd24sd1234springdatajpa1
;

import java.util.Optional;

import org.junit.jupiter.api.Test;
```

```java
import
org.springframework.beans.factory.annotation.
Autowired;
import
org.springframework.boot.test.context.SpringB
ootTest;
import
org.springframework.context.ApplicationContex
t;

import
com.spring.data.jpa.csd24sd1234springdatajpa1
.data.Product;
import
com.spring.data.jpa.csd24sd1234springdatajpa1
.repo.ProductRepository;

@SpringBootTest
class
Csd24sd1234Springdatajpa1ApplicationTests {

    @Autowired
    ApplicationContext context;

    @Test
    void viewProduct() {

        ProductRepository repo =
context.getBean(ProductRepository.class);

        //To insert a record
//      Product product = new Product();
//      product.setProductid(117);
```

```
//      product.setProductname("Oppo
SmartPhone");
//      product.setPrice(35000d);
//      repo.save(product);

        //To fetch a record
//      Optional<Product> productOptional =
repo.findById(110);
//      if (productOptional.isPresent()) {
//          Product result =
productOptional.get();
//          System.out.println(result);
//      }

        //To update a record
        Optional<Product> productOptional =
repo.findById(116);
        if (productOptional.isPresent()) {
            Product result =
productOptional.get();
            System.out.println(result);
            result.setPrice(25000d);
            repo.save(result);
            repo.findAll().forEach(p-
>{System.out.println(p.getProductname() + " &
"+p.getPrice());});

        //To delete a record

        //To fetch all records

        }
    }
```

```
}
```

Output:

1. For Save method, cross check with DB, if new record created with above values.
2. For Get method, check in console output, if the extracted record values are printed

How to name the data members of the model class that replicate the Table in data base:

Database Table Field Name	Model Class of the Entity
id	id
ProductId	productid
Product_Id	productId (select product_Id..)

Plan 2: Connect Restful Client with DB via Spring Application

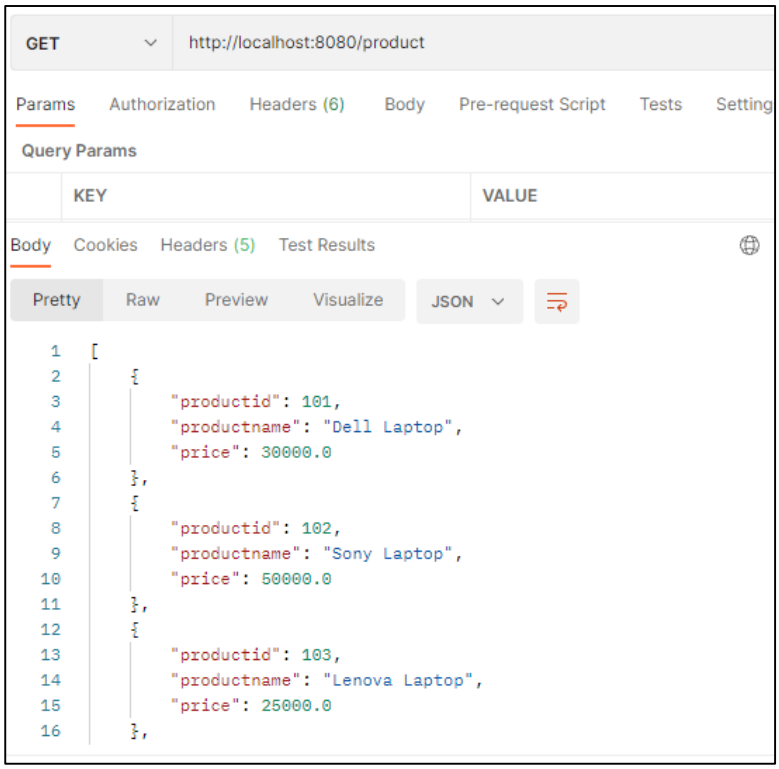
Controller class:

```
package
com.spring.data.jpa.csd24sd1234springdatajpa1
.controller;

import
org.springframework.beans.factory.annotation.
Autowired;
import
org.springframework.web.bind.annotation.GetMa
pping;
import
org.springframework.web.bind.annotation.PostM
apping;
import
org.springframework.web.bind.annotation.Reque
stBody;
import
org.springframework.web.bind.annotation.Reque
stMapping;
```

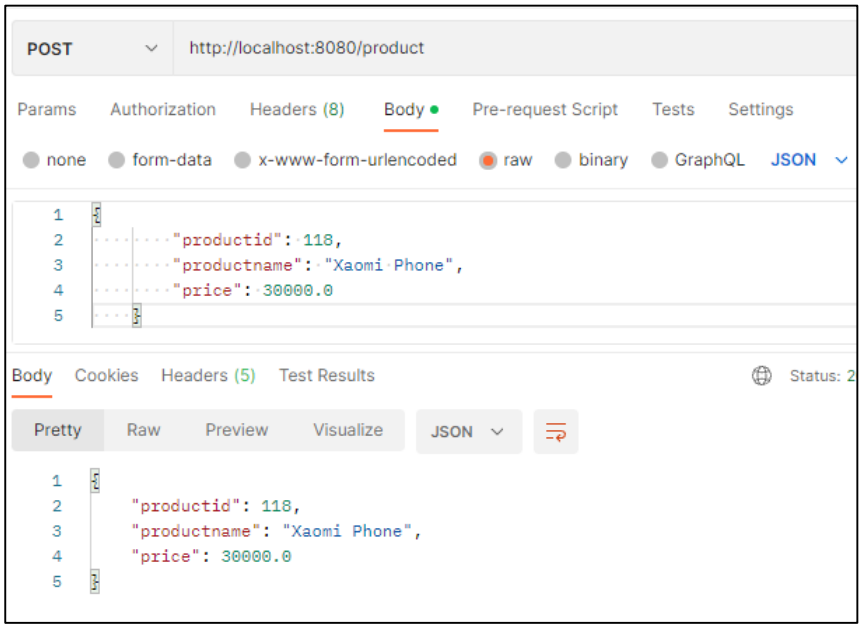
```java
import
org.springframework.web.bind.annotation.RestC
ontroller;

import
com.spring.data.jpa.csd24sd1234springdatajpa1
.data.Product;
import
com.spring.data.jpa.csd24sd1234springdatajpa1
.repo.ProductRepository;

@RestController
@RequestMapping("/product")
public class ProductController {

    @Autowired
    ProductRepository repository;

    @GetMapping
    public Iterable<Product> getProducts(){
        return repository.findAll();
    }

    @PostMapping
    public Product createRecord(@RequestBody
Product product) {
        return repository.save(product);
    }

    //Update a Record
    //Delete a Record
}
```

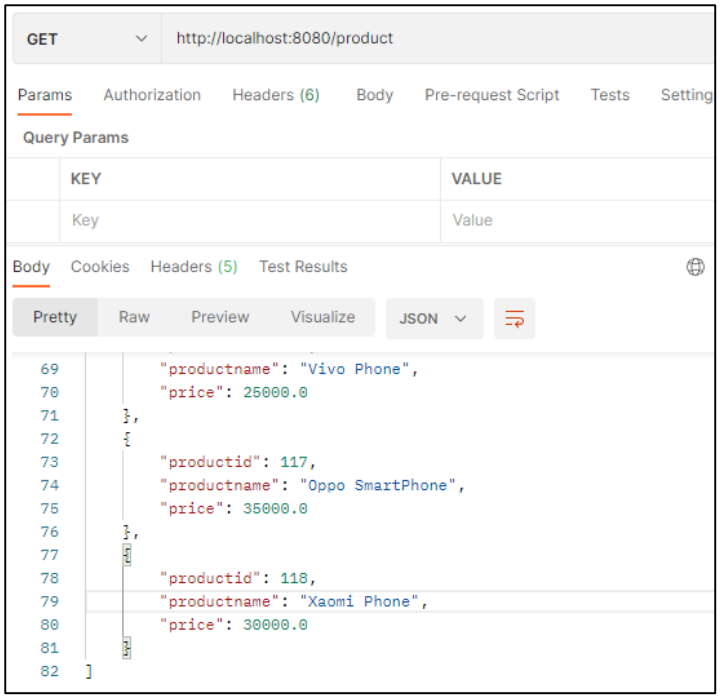
Open Postman tool & give below URL with GET Method

localhost:8080/product



Use POST method with data passed in method body





Also, cross check with DB, if new record created with above values.

Exercise:

Do the rest of methods as indicated in comment section of the program.