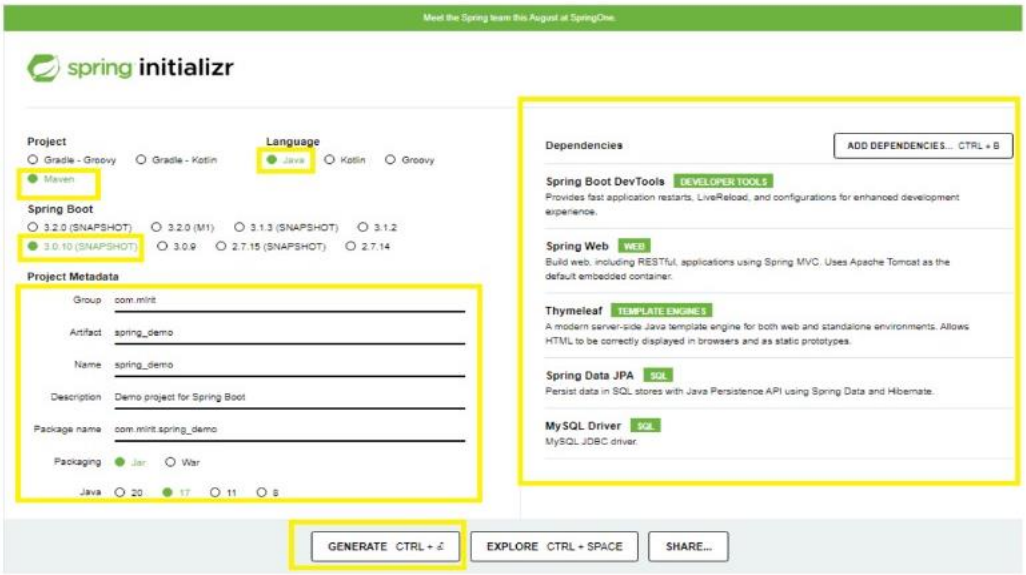
	Exp.No: 12 **Date: 13-06-2023**

Aim: CRUD OPERATIONS USING SPRING BOOT

Write a java program to perform CRUD operations using spring or spring boot. (Update, Delete)

Create a spring project go to the link - https://start.spring.io/, do all the things highlighted and generate, then a “zip” file will be download as soon as you hit the generate button.



Once the download is complete, Extract the zip file Import the project folder into the Eclipse IDE. Implementation of CRUD operations in Springboot

When you import the downloaded spring boot application by default SpringDemoApplication.java is created which consists of the basic template for the application.

SpringDemoApplication.java

```
package com.mlrit.spring_demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringDemoApplication {
public static void main(String[] args) {
SpringApplication.run(SpringDemoApplication.class, args);
}
}
```

After successfully importing the project files, Open the MySQL Workbench and open the connection and create a database “springbootdb” inside that create a table product with attributes id, version, name, price, and product_id.

Then, in the Eclipse IDE you have to Create the package com.mlritm. service. Inside the package, you have to create an interface ProductService and a class ProductServiceImpl.

ProductSercive.java

```
package com.mlritm.service;
import com.mlritm.entity.Product;
public interface ProductService {
Iterable<Product> listAllProducts();
Product getProductById(Integer id);
Product saveProduct(Product product);
void deleteProduct(Integer id);
}
```

ProductServiceImpl.java

```
package com.mlritm.service;
import com.mlritm.entity.Product;
import com.mlritm.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
/**
* Product service implement.
*/
@Service
public class ProductServiceImpl implements ProductService {
@Autowired
private ProductRepository productRepository;
@Override
public Iterable<Product> listAllProducts() {
return productRepository.findAll();
}
@Override
public Product getProductById(Integer id) {
return productRepository.findById(id).get();
}
@Override
public Product saveProduct(Product product) {
return productRepository.save(product);
}
@Override
public void deleteProduct(Integer id) {
productRepository.deleteById(id);
}
}
```

After that you have to Create the package com.mlritm.controller. Inside the package, you have to create the classes IndexController and ProductController. This class deals with RESTful APIs for CRUD operations.

IndexController.java

```
package com.mlritm.controller;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.GetMapping;

@RestController

public class IndexController {

@GetMapping("/")
```

```
String index() {

return "index";

}}
```

ProductController.java

```
package com.mlritm.controller;
import com.mlritm.entity.Product;
import com.mlritm.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PutMapping;
/**
* Product controller.
*/
@RestController("/products")
public class ProductController {
@Autowired
private ProductService productService;
@GetMapping("/")
public String list(Model model) {
model.addAttribute("products", productService.listAllProducts());
System.out.println("Returning products:");
return "products";
}
@GetMapping("/{id}")
public String showProduct(@PathVariable Integer id, Model model) {
model.addAttribute("product", productService.getProductById(id));
return "productshow";
}
// code to modify the product
@PutMapping("/edit/{id}")
public String edit(@PathVariable Integer id, Model model) {
model.addAttribute("product", productService.getProductById(id));
return "productform";
}
@RequestMapping("product/new")
public String newProduct(Model model) {
model.addAttribute("product", new Product());
return "productform";
}
@RequestMapping(value = "product", method = RequestMethod.POST)
public String saveProduct(Product product) {
productService.saveProduct(product);
return "redirect:/product/" + product.getId();
}
@DeleteMapping("/{id}")
public String delete(@PathVariable Integer id) {
productService.deleteProduct(id);
return "redirect:/products";
}
```

```
}
```

Then, you must create the package com.mlritm.repository inside the package you have to create an EmployeeRepository.java file. ProductRepository interface extends the JPARepository interface which is used to perform the CRUD operations in Springboot.

ProductRepository.java

```
package com.mlritm.repository;
import com.mlritm.entity.Product;
import org.springframework.data.jpa.repository.*;
public interface ProductRepository extends JpaRepository<Product, Integer> {
}
```

Then, you must Create the package com.mlritm.entity inside the package you have to create the class Product. This class retrieves the information from the database and displays it to the user.

Product.java

```
package com.mlritm.entity;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Version;
import java.math.BigDecimal;
/**
* Product entity.
*/
@Entity
public class Product {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer id;
@Version
private Integer version;
private String productId;
private String name;
private BigDecimal price;
public Product() {
}
public Integer getId() {
return id;
}
public void setId(Integer id) {
this.id = id;
}
public Integer getVersion() {
return version;
}
public void setVersion(Integer version) {
this.version = version;
}
public String getProductId() {
return productId;
}
public void setProductId(String productId) {
this.productId = productId;
}
public String getName() {
return name;
}
```

```
public void setName(String name) {
this.name = name;
}
public BigDecimal getPrice() {
return price;
}
public void setPrice(BigDecimal price) {
this.price = price;
}
}
```

After that, you have to create the index.html page inside the folder src/main/resources. This file is created to create a User Interface to perform CRUD operations in Springboot and MYSQL.

Index.html:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootswatch/4.5.2/cosmo/bootstrap.min.css" />
<script src= "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" ></script>
</head>
<body>
<div>
<h2 >Spring Boot Crud System - Product</h2>
<table class="table">
<tr>
<td>
<div align = "left" >
<h3><a th:href="@{'/new'}">Add new</a></h3>
</div>
</td>
</tr>
<tr>
<div class="col-sm-5" align = "center">
<div class="panel-body" align = "center" >
<thead class="thead-dark">
<tr>
<th>ID</th>
<th>Version</th>
<th>Name</th>
<th>Price</th>
<th>Product ID</th>
<th>Edit</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr th:each="product : ${listproduct}">
<td th:text="${product.id}">ID</td>
<td th:text="${product.version}">Version</td>
<td th:text="${product.name}">Name</td>
<td th:text="${product.price}">Price</td>
<td th:text="${product.product_id}">Product ID</td>
<td>
<a th:href="@{'/edit/' + ${product.id}}">Edit</a>
</td>
<td>
```

```
<a th:href="@{'/delete/' + ${product.id}}">Delete</a>
</td>
</tr>
</tbody>
</div>
</div>
</tr>
</tbody>
</table>
<div>
</body>
</html>
```

Inside the src/main/resources folder, create another HTML file new.html. This new file is created to open a new page to create a new user.

new.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8" />
<title>Create New Product</title>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootswatch/4.5.2/cosmo/bootstrap.min.css" />
<script src= "https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" ></script>
</head>
<body>
<div align="center">
<h1>Create New Employee</h1>
<br />
<div class="col-sm-4">
<form action="#" th:action="@{/save}" th:object="${employee}" method="post">
<div alight="left">
<tr>
<label class="form-label" >Employee Name</label>
<td>
<input type="hidden" th:field="*{id}" />
<input type="text" th:field="*{ename}" class="form-control" placeholder="Emp Name" />
</td>
</tr>
</div>
<div alight="left">
<tr>
<label class="form-label" >mobile</label>
<td>
<input type="text" th:field="*{mobile}" class="form-control" placeholder="mobile" />
</td>
</tr>
</div>
<div alight="left">
<tr>
<label class="form-label" >salary</label>
<td><input type="text" th:field="*{salary}" class="form-control" placeholder="salary" /></td>
</tr>
</div><br>
<tr>
<td colspan="2"><button type="submit" class="btn btn-info">Save</button> </td>
</tr>
```

```
</form>
</div>
</body>
</html>
```

After that, we have to set the database path and project configuration in the application.properties file located in the src/main/resources folder.

```
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb
server.error.whitelabel.enabled=false
server.port=7070
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.open-in-view=false
spring.thymeleaf.cache=false
api.base.path = http://localhost:7070/
```

To run our application, go to the browser and check the site localhost:7070. Our final application has the features to read the product details, create new product data, update the product details and delete the product data.

Output: