DEPARTMENT OF CSE
COURSE CODE: 23SDCS12A / 23SDCS12R
FULL STACK APPLICATION DEVELOPMENT

Date of the Session: __/__/_____ **Time of The Session:_____to_____**

LAB – 8 ➔ Spring Boot with Rest API and CRUD Operations

Prerequisites:
General Idea on Spring Boot MVC and Form Handling
General Idea on Spring Data JPA

Exercise:
Develop a Spring Boot web application to manage a list of products in a warehouse. The application should handle CRUD operations to manage product details such as Product ID, Name, Description, Price, and Quantity. The application should include features to add new products, display a list of all products, update existing product details, and delete products from the database. Use Spring Web MVC for handling HTTP requests, Spring Data JPA for database interactions. Ensure the application is configured to connect to a MySQL/PostgreSQL database and implement both setter-based or constructor-based dependency injections to manage service and repository layers effectively.

❖ Watch The Video And Do In Eclipse Workspace

8a https://youtu.be/qOR8u_hobTA?si=orkN06GkavNTUz4H

8b https://youtu.be/t2ftcy0T5DU?si=EEAfn4kvPGGDoW6t

application.properties

```
spring.application.name=ex8
server.port=8081
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/fsaddemo
spring.datasource.username=root
spring.datasource.password=Root@123
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
```

Product.java

```
package com.klu;

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

@Entity
public class Product {
   @Id
   int id;
   String name;

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   @Override
   public String toString() {
      return "Product [id=" + id + ", name=" + name + "]";
   }
}
```

ProductRepo.java

```
package com.klu;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepo extends JpaRepository<Product, Integer> {

}
```

Service.java

```
package com.klu;

import org.springframework.beans.factory.annotation.Autowired;
import java.util.List;

@org.springframework.stereotype.Service
public class Service {

   @Autowired
   private ProductRepo repo1;

   public String insertData(Product product) {
      repo1.save(product);
      return "Inserted Successfully";
   }

   public String updateData(Product product) {
      if (repo1.findById(product.getId()) != null)
         repo1.delete(product);
         repo1.save(product);
         return "Updated Successfully";
      }
```

```
    public String deleteData(int id) {
        repo1.delete (repo1.findById(id) .get());
            repo1.deleteById(id);
            return "Deleted Successfully";
    }

    public List<Product> retrieveData() {
        return repo1.findAll();
    }
}
```

Appcontroller.java

```
package com.klu;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class Appcontroller {

    @Autowired
    Service s;

    @PostMapping("/product")
    public String insertProduct(@RequestBody Product product) {
        return s.insertData(product);
    }
```

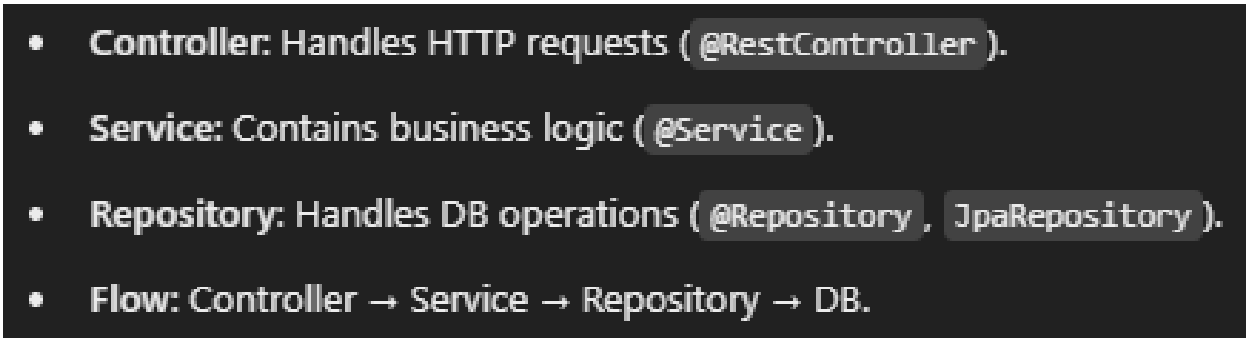
```
    @PutMapping("/product")
    public String updateProduct(@RequestBody Product product) {
        return s.updateData(product);
    }

    @DeleteMapping("/product/{id}")
    public String deleteProduct(@PathVariable int id) {
        return s.deleteData(id);
    }

    @GetMapping("/product")
    public List<Product> retrieveProduct() {
        return s.retrieveData();
    }
}
```

Ex8Application.java

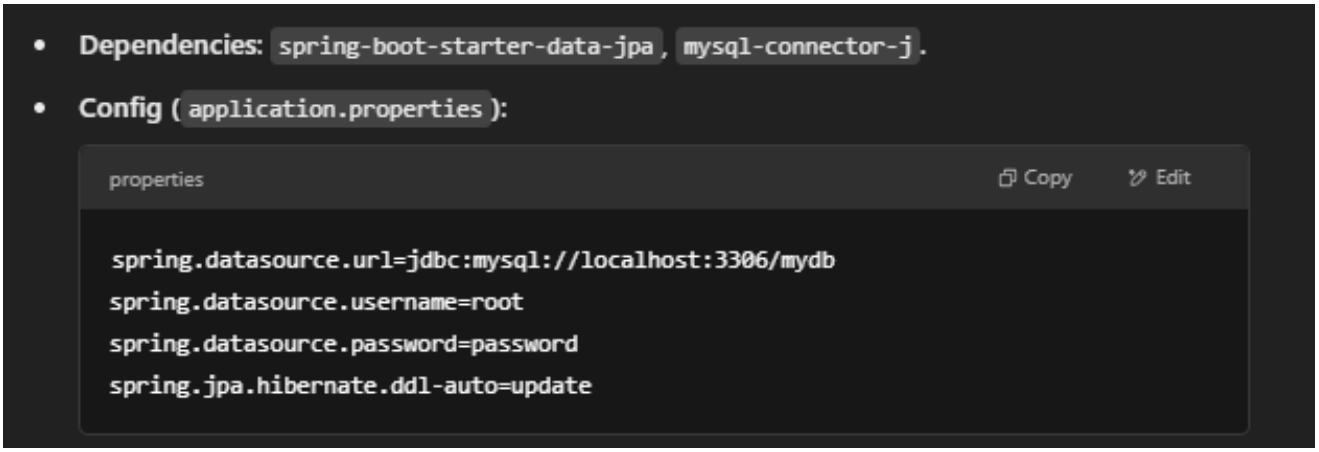
```
package com.klu;

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

@SpringBootApplication
public class Ex8Application {

    public static void main(String[] args) {
        SpringApplication.run(Ex8Application.class, args);
    }
}
```

VIVA QUESTIONS:

1. Can you explain the role of each layer (Controller, Service, Repository) in a Spring Boot MVC application, especially in the context of CRUD operations?



2. How would you configure and connect a Spring Boot application to a relational database, and what dependencies are necessary for CRUD operations?



3. Describe how you would create and map a JPA entity for a table in the database. How does this mapping support CRUD operations?

java Copy Edit

```java
@Entity
@Table(name = "users")
public class User {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;
}
```

- `@Entity` → Marks a DB table.
- `@Id` → Primary key.
- `JpaRepository<User, Long>` → Auto CRUD.

4. How do you handle data validation in a Spring Boot CRUD application before saving data to the database? Can you give examples of annotations used for validation?

java Copy Edit

```java
@NotBlank(message = "Name is required")
private String name;

@Email(message = "Invalid email")
private String email;
```

- **Common Annotations:** `@NotBlank`, `@Email`, `@Size`.
- **Validate in Controller:** `@Valid @RequestBody User user`.

5. What is the purpose of @Transactional in Spring Boot, and how does it ensure data consistency during CRUD operations?

- Ensures **atomicity & rollback** on failure.
- Handles multiple queries as one transaction.
- Example:

java Copy Edit

```java
@Transactional
public void updateUser(Long id, String email) {
    User user = repo.findById(id).orElseThrow();
    user.setEmail(email);
    repo.save(user);
}
```

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured _____out of 50 Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation: