IFT LAB4-REPORT

KRITHIKA H PADMASHALI

01

230958002

**PRACTISE QUESTION**:

Objectives:

1. Set up a MySQL database and configure it for integration with Spring Boot.

2. Implement a Spring Boot project with RESTful endpoints interacting with a MySQL database.

3. Design a complete Spring Boot application with a custom database schema and CRUD operations.

4. Develop additional layers such as Service and Exception Handling to enhance the functionality and reliability of the application.

**Solution:**

**Step 1: Setup MySQL Database**

1. Install MySQL (if not already done).

2. Start MySQL and create a new database: CREATE DATABASE springboot\_db;

**Step 2: Create a Spring Boot Project**

1. Using Spring Initializer (Recommended):

o Go to Spring Initializer.

o Select:

▪ Project: Maven or Gradle.

▪ Dependencies:

▪ Spring Web

▪ Spring Data JPA

▪ MySQL Driver

o Generate the project as a .zip file and extract it.

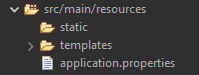
1. Using Your IDE: o Create a new Maven/Gradle project.

o Add dependencies to the pom.xml or build.gradle file.

**Step 3: Add Dependencies**

Add all the afore mentioned dependencies to your pom.xml file

**Step 4: Configure Application Properties**



spring.application.name=FIS01

# --- Server Configuration ---

server.port=8080

# --- Thymeleaf Configuration ---

spring.thymeleaf.cache=false

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML5

# --- Database Configuration ---

spring.datasource.url=jdbc:mysql://localhost:3306/springboot\_db

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# --- JPA Configuration ---

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

spring.jpa.show-sql=true

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

# --- Validation Configuration ---

spring.messages.basename=messages

spring.messages.encoding=UTF-8

# --- Logging ---

logging.level.org.hibernate.SQL=DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

**Step 5: Create Entity Class**

package com.example.FIS01.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

*@Entity*

*@Table*(name = "`user`")

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

*@NotBlank*(message = "Name is required")

*@Size*(max = 50, message = "Name must not exceed 50 characters")

private String name;

*@NotBlank*(message = "Email is required")

*@Email*(message = "Invalid email address")

private String email;

// Getters and setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**Step 6: Create Repository Interface**

package com.example.FIS01.repository;

import com.example.FIS01.entity.User;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**Step 7: Create a Service Layer**

Add business logic by creating a service class.

package com.example.FIS01.service;

import com.example.FIS01.entity.User;

import com.example.FIS01.repository.UserRepository;

import jakarta.validation.Valid;

import java.util.List;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.stereotype.Service;

*@Service*

public class UserService {

*@Autowired*

private UserRepository userRepository;

public Page<User> getPaginatedUsers(int page, int pageSize) {

return userRepository.findAll(PageRequest.*of*(page, pageSize));

}

public List<User> getAllUsers() {

// **TODO** Auto-generated method stub

return userRepository.findAll();

}

public void saveUser(*@Valid* User user) {

userRepository.save(user);}}

**Step 8: Create a Controller**

Expose REST endpoints for interacting with the database.

package com.example.FIS01.controller;

import com.example.FIS01.service.UserService;

import jakarta.validation.Valid;

import com.example.FIS01.entity.User;

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

import org.springframework.data.domain.Page;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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

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

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

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

*@Controller*

public class UserController {

*@Autowired*

private UserService userService;

*@GetMapping*("/")

public String home() {

return "home";

}

*@GetMapping*("/users")

public String getUsers(

*@RequestParam*(defaultValue = "1") int page,

Model model) {

int pageSize = 8; // Number of users per page

Page<User> userPage = userService.getPaginatedUsers(page - 1, pageSize);

model.addAttribute("users", userPage.getContent());

model.addAttribute("currentPage", page);

model.addAttribute("totalPages", userPage.getTotalPages());

model.addAttribute("pageTitle", "User List");

return "users";

}

*@GetMapping*("/allusers")

public String listUsers(Model model) {

model.addAttribute("pageTitle", "User List");

model.addAttribute("users", userService.getAllUsers());

return "userdetails";

}

*@GetMapping*("/new")

public String showCreateForm(Model model) {

model.addAttribute("pageTitle", "Add User");

model.addAttribute("user", new User());

return "userform";

}

*@PostMapping*("save")

public String saveUser(*@Valid* *@ModelAttribute*("user") User user, BindingResult result, Model model) {

if (result.hasErrors()) {

model.addAttribute("pageTitle", "Add User");

return "userform";

}

userService.saveUser(user);

return "redirect:/users";

}

}

Step 9: Run the Application

1. Run the main application class (DemoApplication.java):

package com.example.FIS01;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Fis01Application {

public static void main(String[] args) {

SpringApplication.run(Fis01Application.class, args);

}

}

1. Verify that the application starts without errors.

**Step 10: Test the Application**

1. Use Postman or cURL to test the API:

GET: http://localhost:8080/users

POST: <http://locahost:8080/users>

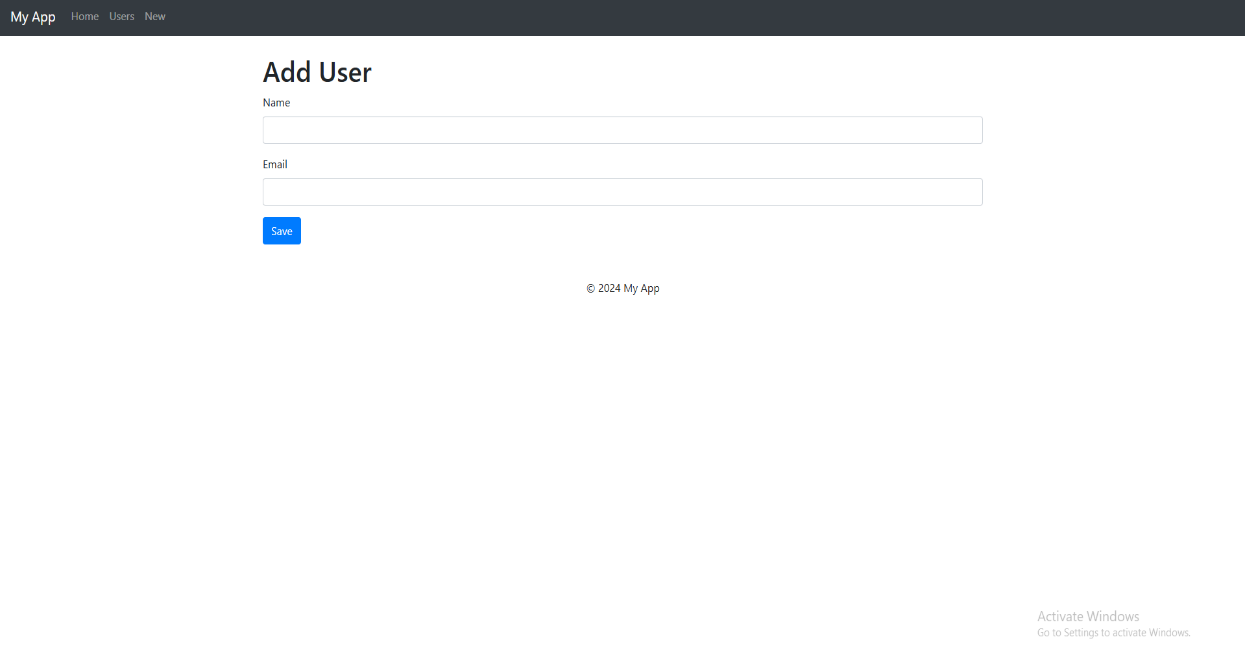
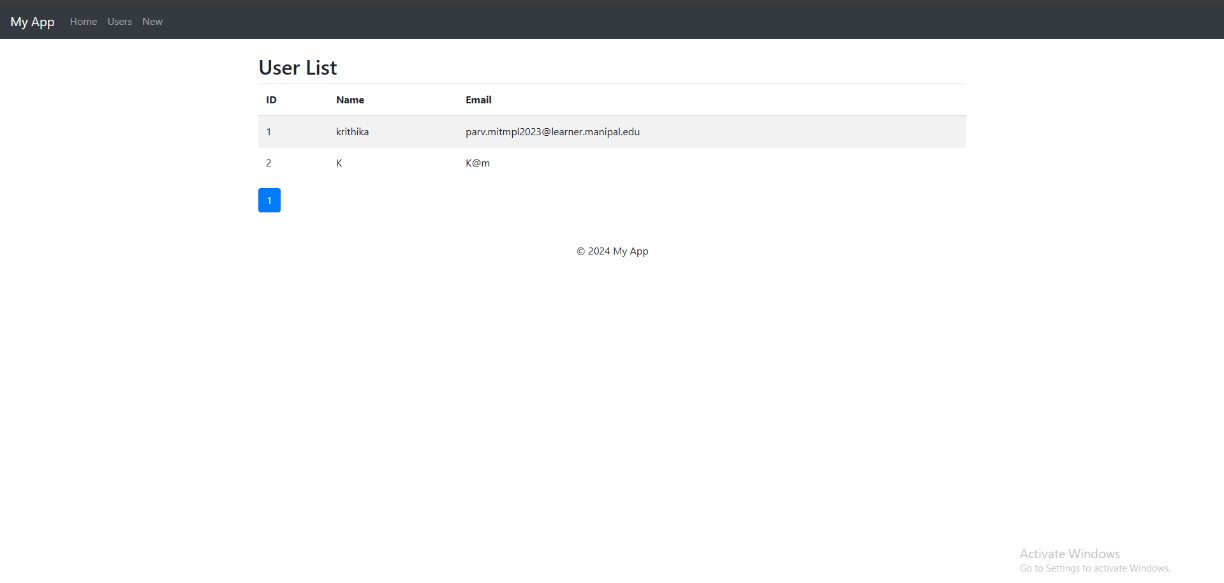
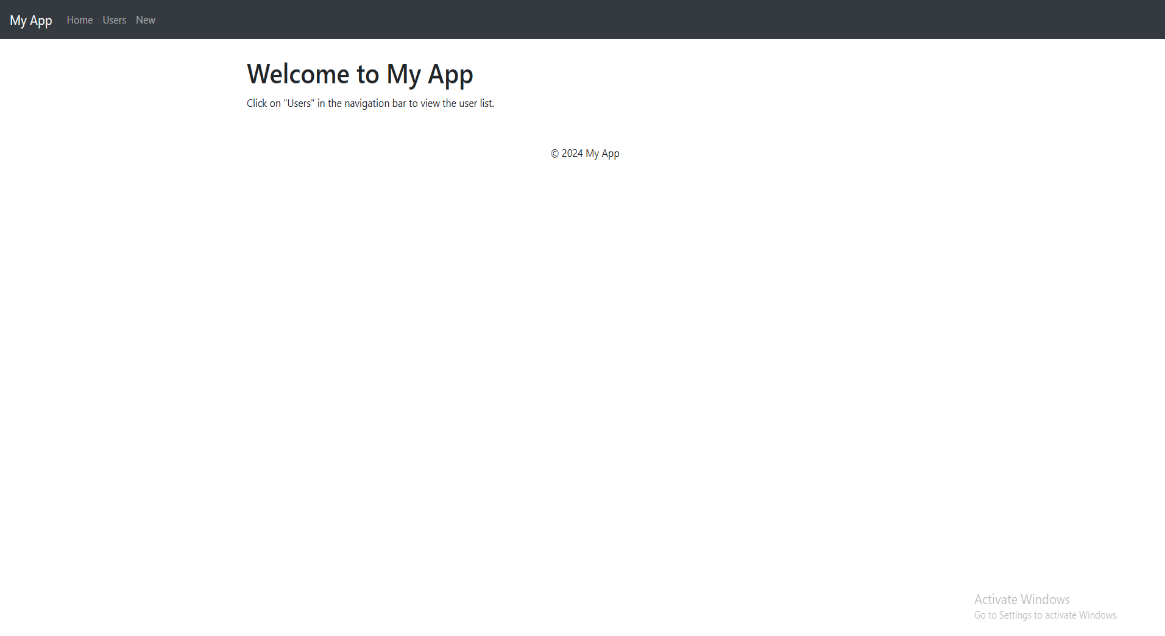
1. Check the MySQL database to confirm data persistence:



select \* from springboot\_db.user;

**Step 11: View it on the web**

Browse for localhost 8080 on your preferred browser



**LAB QUESTIONS:**

1)Establish Spring Boot MySQL Connection with CustomerDetail table

2)Establish Spring Boot MySQL Connection with CustomerIdentification table

3)Establish Spring Boot MySQL Connection with CustomerContactInformation table

4)Establish Spring Boot MySQL Connection with CustomerProofOfId table