PWC Module 3: Implement Frameworks the DevOps Way

**Discuss Maven Project**

1. Advantages of using maven projects
2. Discuss maven dependencies handling and the **pom** file
3. Demonstrate how dependencies are stored (remote and local) repositories
4. Add spring framework dependencies

Ex:

Adding “Spring-Context” to our project

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

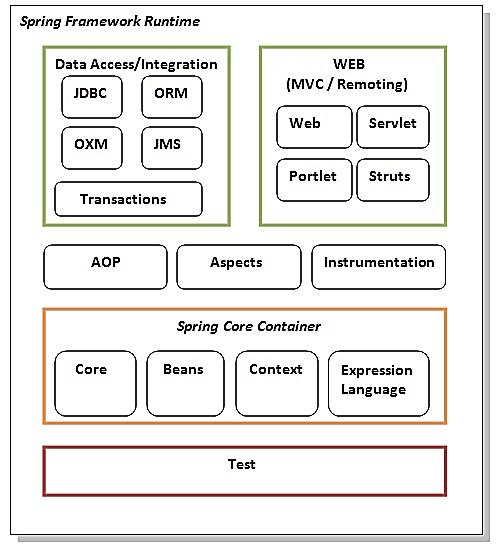
<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.0.11</version>

</dependency>

**Understand Spring basics, including its framework architecture**



1. Using ApplicationContext and/or BeanFactory with XML Configuration

Understanding Dependency Injection:

When we have a class that depends on multiple classes, it is often recommended to implement “loose coupling” where when we would want to change the class that we are pointing to, we will not change the java code, but rather modify a separate file (like an XML file).

Example:

[Tight Coupling]

Philippines.java

package com.sas.spring1;

public class Philippines {

void greet() {

System.out.println("Kumusta kababayan");

}

}

United States.java

package com.sas.spring1;

public class UnitedStates {

void greet() {

System.out.println("Hello There");

}

}

App.java

package com.sas.spring1;

public class App

{

public static void main( String[] args )

{

Philippines ph = new Philippines();

ph.greet();

}

}

If I want to run the greet from the United States class, I would need to modify my java code, recompile it and launch it. This is an example of Tight Coupling.

Now we will use Spring Class (ApplicationContext) to demonstrate loose coupling.

1. Create an interface that will call the greet method

Country.java (interface)

package com.sas.spring1;

public interface Country {

void greet();

}

1. Modify the Philippines and United States classes to implement the Country interface

package com.sas.spring1;

public class Philippines **implements Country** {

public void greet() {

System.out.println("Kumusta kababayan");

}

}

package com.sas.spring1;

public class UnitedStates implements Country{

public void greet() {

System.out.println("Hello There");

}

}

1. Then create an XML file to hold the definitions and configurations

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<bean id="country" class="com.sas.spring1.UnitedStates"></bean>

</beans>

1. Then our main class will be like this

App.java

package com.sas.spring1;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext context = new ClassPathXmlApplicationContext("Spring.xml");

Country c = (Country)context.getBean("country");

c.greet();

}

}

Where if I want to modify the output to display the greeting in Philippines, I can modify the xml file and it would not need a recompile of the project for the changes to take effect. Try modifying your xml file to display the output from the other class.

1. Using ApplicationContext / BeanFactory with Annotations

Instead of having all the configuration in the xml file, we can set each class as the components as well

Philippines.java

package com.sas.spring1;

**import org.springframework.stereotype.Component;**

**@Component**

public class Philippines implements Country {

public void greet() {

System.out.println("Kumusta kababayan");

}

}

United States.java

package com.sas.spring1;

import org.springframework.stereotype.Component;

@Component

public class UnitedStates implements Country{

public void greet() {

System.out.println("Hello There");

}

}

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

**<context:component-scan base-package="com.sas.spring1"></context:component-scan>**

</beans>

In using the Components, we will be using the **lowercase version of the class names**

package com.sas.spring1;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext ac = new ClassPathXmlApplicationContext("Spring.xml");

Country c = (Country)ac.getBean("**philippines**");

c.greet();

}

}

1. Bean Properties
2. Create a new class

City.java

package com.sas.spring1;

public class City {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "City [name=" + name + "]";

}

}

\*note:

the getters and setters can be automatically generated by rclick🡪source🡪generate getters and setters

the toString method can be automatically generated by rclick🡪source🡪generate toString Method

1. We can then modify the xml file to add annotation or xml configuration. IN this example, we will do XML configuration

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.sas.spring1"></context:component-scan>

**<bean id="city" class="com.sas.spring1.City" >**

**<property name="name" value="Manila"></property>**

**</bean>**

</beans>

And to use it in our Main App

App.java

package com.sas.spring1;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext context = new ClassPathXmlApplicationContext("Spring.xml");

/\*

\* Country c = (Country)context.getBean("philippines"); c.greet();

\*/

City newcity = (City)context.getBean("city");

System.out.println(newcity);

}

}

**Build RESTful web services**

Build a simple RESTful API that returns a string.

1. Demo using Spring Initializr
2. In eclipse, install the Spring tools plugin

Help🡪Eclipse Marketplace🡪search for “Spring Tools”

A screenshot of a computer

Description automatically generated

https://start.spring.io/

1. Create Spring Boot project from Eclipse

File🡪New🡪Spring🡪New Spring Starter Project

Configuration:

|  |  |
| --- | --- |
| Project | Maven |
| Language | Java |
| Spring Boot | 3.2.0 |
| Group | com.restproject |
| Artifact | Rest1 |
| Name | Rest1 |
| Description |  |
| Package Name | com.restproject.Rest1 |
| Packaging | Jar |
| Java Version | 20 |
| Dependencies | Spring Web |

1. Discuss the Spring Boot Application Hierarchy
2. Try running it as a Spring Boot Application
3. Add a new controller class.

Note: You might not see any results yet so we will construct a Controller class. All controller class names should be appended with the word Controller. Example: “HelloWorldController”

HelloWorldController.java

package com.example.demo;

import org.springframework.stereotype.Controller;

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

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

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

//@Controller

//@ResponseBody

//or simply

@RestController

public class HelloWorldController {

// defined http get method

// can also work with a defined url

// http://localhost:8080/hello-world

@GetMapping("/hello-world")

public String helloWorld() {

return "Hello World";

}

}

1. Test it on the browser http://localhost:8080/hello-world

Build a RESTful API that returns Java Bean / JSON

1. Create a class that represents the object you need to return as a response

Employee.java

package com.example.demo;

public class Employee {

private String firstName;

private String lastName;

//rclick->source->generate constructor from fields

public Employee(String firstName, String lastName) {

super();

this.firstName = firstName;

this.lastName = lastName;

}

//rclick->source->generate getters and setters

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

}

1. Create a controller class

EmployeeController.java

package com.example.demo;

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

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

@RestController

public class EmployeeController {

//http://localhost:8080/employee

@GetMapping("/employee")

public Employee getEmployee() {

return new Employee("john", "doe");

}

}

1. Run the REST API and try accessing the endpoint http://localhost:8080/employee

Build a RESTful API that returns a list

1. From the previous activity, modify the EmployeeController.java

package com.example.demo;

import java.util.ArrayList;

import java.util.List;

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

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

@RestController

public class EmployeeController {

//http://localhost:8080/employee

@GetMapping("/employee")

public Employee getEmployee() {

return new Employee("john", "doe");

}

**//http://localhost:8080/employees**

**@GetMapping("/employees")**

**public List<Employee> getEmployees(){**

**List<Employee> employees = new ArrayList<>();**

**employees.add(new Employee("Kevin", "Sanchez"));**

**employees.add(new Employee("Mike", "Morey"));**

**employees.add(new Employee("Lisa", "Downey"));**

**return employees;**

**}**

}

Build a RESTful API with Path Variable (@PathVariable)

Note: Path Variables aka Path Parameters are values that are submitted from a URL

1. Modify the EmployeeController.java and add another method

package com.example.demo;

…

@RestController

public class EmployeeController {

…

**//http://localhost:8080/employee/steph/curri**

**//to bind URI template variable to method variable, use @PathVariable**

**@GetMapping("/employee/{firstName}/{lastName}")**

**public Employee employeePathVariable(**

**@PathVariable("firstName") String firstName,**

**@PathVariable("lastName")String lastName**

**)**

**{**

**return new Employee(firstName,lastName);**

**}**

}

Build a RESTful API with Request Parameter (@RequestParam)

1. Modify the EmployeeController.java to add a new method

package com.example.demo;

…

@RestController

public class EmployeeController {

…

**//Rest API Endpoint to handle query parameter**

**//http://localhost:8080/employee/query?firstName=Kevin&lastName=Turney**

**@GetMapping("/employee/query")**

**public Employee employeeRequestParameter(**

**@RequestParam(name = "firstName") String firstName,**

**@RequestParam(name = "lastName") String lastName**

**)**

**{**

**return new Employee(firstName,lastName);**

**}**

}

**Spring Boot + Hibernate + MySQL CRUD REST API**

A diagram of a project

Description automatically generated

1. Create a database in MySQL

Database Name: Ems

Table Name:

1. Create Spring Boot Project

File🡪New🡪Spring🡪New Spring Starter Project

Configuration:

|  |  |
| --- | --- |
| Project | Maven |
| Language | Java |
| Spring Boot | 3.2.0 |
| Group | com.restproject |
| Artifact | Rest1 |
| Name | Rest1 |
| Description |  |
| Package Name | com.restproject.Rest1 |
| Packaging | Jar |
| Java Version | 20 |
| Dependencies | Spring Web  Spring Data JPA  MySQL Driver  Lombok |

1. Create a packaging structure.

A screenshot of a computer program

Description automatically generated

1. Modify the application.properties file

# Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/ems

spring.datasource.username=john

spring.datasource.password=123

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

# spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver #or you can just leave this and the preceding line active

# JPA Configuration

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

spring.jpa.hibernate.ddl-auto=update

# spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration

1. If, problems occurs, try commenting out this dependency in pom.xml (uncomment later after first successful build)

…

<!--<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>-->

…

1. Create a JPA entity

Note:

* We will be using Lombok library (@Data) which will reduce boilerplate code like getters/setters, constructors, toString, and other required methods for java classes.
* We will also be using @Entity from the javax persistence package (if spring boot 2.7 or older) or Jakarta persistence package (if spring boot 3) to make this class a jpa entity
* Add contructor for the fields (except the autogenerated id)
* Add getter and setters (except for the autogenerated id)

Rclick the model package🡪new class (“Employee”)

package com.example.demo.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

import lombok.Data;

@Data

@Entity

@Table(name="employees")

public class Employee {

//define primary key

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

//set column definition

@Column(name="first\_name")

private String firstName;

@Column(name="last\_name")

private String lastName;

@Column(name="email")

private String email;

// default (empty) constructor

// rclick Source->generate constructor using fields (uncheck all fields)

public Employee() {

super();

}

// constructor that uses params

// rclick Source->generate constructor using fields (check all fields)

public Employee(String firstName, String lastName, String email) {

super();

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

}

// create getters and setters for all our properties

// rclick Source-->generate getters and setters (check all fields

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

1. Create a NotFound Error Exception

Rclick Exception Package🡪new class (“ResourceNotFoundException”)

ResourceNotFoundException.java

package com.example.demo.exception;

import org.springframework.http.HttpStatus;

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

@ResponseStatus(value=HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException{

private static final long serialVersionUID = 1L;

private String resourceName;

private String fieldName;

private Object fieldValue;

//generate constructors for the fields

public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {

super(String.format("%s not found with %s : %s",resourceName,fieldName,fieldValue));

this.resourceName = resourceName;

this.fieldName = fieldName;

this.fieldValue = fieldValue;

}

//generate getters for the properties

public String getResourceName() {

return resourceName;

}

public String getFieldName() {

return fieldName;

}

public Object getFieldValue() {

return fieldValue;

}

}

1. Create EmployeeRepository Interface

Rclick Repository Package🡪new Interface (“EmployeeRepository”)

package com.example.demo.repository;

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

import com.example.demo.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

1. Create EmployeeService Interface

Rclick Service Package🡪new Interface (“EmployeeService”)

package com.example.demo.service;

import com.example.demo.model.Employee;

public interface EmployeeService {

Employee saveEmployee(Employee employee);

}

1. Create a class that implements the EmployeeService Interface

Rclick Service.impl Package🡪new Class (“EmployeeServiceImpl”)

package com.example.demo.service.impl;

import org.springframework.stereotype.Service;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

// dependency injection from the repository class

private EmployeeRepository employeeRepository;

// create constructor for this class

public EmployeeServiceImpl(EmployeeRepository employeeRepository) {

super();

this.employeeRepository = employeeRepository;

}

@Override

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

}

Note: You can easily implement unimplemented methods from the interface that is being extended by hovering the mouse over the class name then clicking “Add unimplemented methods”

A screenshot of a computer

Description automatically generated

**Adding new records**

1. Create a controller class that will provide mapping of API endpoints

Rclick Controller package🡪new class (“EmployeeController”)

EmployeeController.java

package com.example.demo.controller;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

super();

this.employeeService = employeeService;

}

// create API Endpoints

// save new record

@PostMapping()

public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee){

return new ResponseEntity<Employee>(

employeeService.saveEmployee(employee),

HttpStatus.CREATED

);

}

}

1. Run the project and try sending JSON Post using PostMan

A screenshot of a computer program

Description automatically generated

**Getting all records**

1. Modify the EmployeeService.java class to add new service method

EmployeeService.java

package com.example.demo.service;

import java.util.List;

import com.example.demo.model.Employee;

public interface EmployeeService {

Employee saveEmployee(Employee employee);

List<Employee> getAllEmployees();

}

1. Modify the EmployeeServiceImpl.java to add the unimplemented method

A screenshot of a computer

Description automatically generated

package com.example.demo.service.impl;

**import java.util.List;**

import org.springframework.stereotype.Service;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

// dependency injection from the repository class

private EmployeeRepository employeeRepository;

// create constructor for this class

public EmployeeServiceImpl(EmployeeRepository employeeRepository) {

super();

this.employeeRepository = employeeRepository;

}

@Override

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

**@Override**

**public List<Employee> getAllEmployees() {**

**// the code inside this method had been manually modified**

**return employeeRepository.findAll();**

**}**

}

1. Modify the EmployeeController.java to add a RESTFul API Endpoint

EmployeeController.java

package com.example.demo.controller;

**import java.util.List;**

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import com.example.demo.service.impl.EmployeeServiceImpl;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

super();

this.employeeService = employeeService;

}

@GetMapping("/test")

public String test() {

return "controller ok";

}

// API Endpoints for CRUD operations

// save new record

@PostMapping("/new")

public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee){

return new ResponseEntity<Employee>(

employeeService.saveEmployee(employee),

HttpStatus.CREATED

);

}

**// get all employees**

**@GetMapping**

**public List<Employee> getAllEmployee(){**

**return employeeService.getAllEmployees();**

**}**

}

1. Launch/Relaunch App then test with PostMan

A screenshot of a computer

Description automatically generated

**Get record by EmployeeID**

1. Modify the EmployeeService.java to add a method that handles getting record by id.

EmployeeService.java

package com.example.demo.service;

import java.util.List;

import com.example.demo.model.Employee;

public interface EmployeeService {

Employee saveEmployee(Employee employee);

List<Employee> getAllEmployees();

Employee getEmployeeById(long id);

}

1. Modify the EmployeeServiceImpl to implement the service methods

EmployeeServiceImpl.java

package com.example.demo.service.impl;

import java.util.List;

import java.util.Optional;

import org.springframework.stereotype.Service;

import com.example.demo.exception.ResourceNotFoundException;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

// dependency injection from the repository class

private EmployeeRepository employeeRepository;

// create constructor for this class

public EmployeeServiceImpl(EmployeeRepository employeeRepository) {

super();

this.employeeRepository = employeeRepository;

}

@Override

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

@Override

public List<Employee> getAllEmployees() {

// the code inside this method had been manually modified

return employeeRepository.findAll();

}

**@Override**

**public Employee getEmployeeById(long id) {**

**// Optional<Employee> employee = employeeRepository.findById(id);**

**// if(employee.isPresent()) {**

**// return employee.get();**

**// }else {**

**// throw new ResourceNotFoundException("Employee", "Id", id);**

**// }**

**// or shorter by using lambda**

**return employeeRepository.findById(id).orElseThrow(**

**() -> new ResourceNotFoundException("Employee", "Id", id)**

**);**

**}**

@Override

public Employee updateEmployee(Employee employee, long id) {

// TODO Auto-generated method stub

return null;

}

}

1. Modify EmployeesController.java to add a REST API endpoint to call the findbyid method in our service

EmployeesController.java

package com.example.demo.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

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

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import com.example.demo.service.impl.EmployeeServiceImpl;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

super();

this.employeeService = employeeService;

}

@GetMapping("/test")

public String test() {

return "controller ok";

}

// API Endpoints for CRUD operations

// save new record

@PostMapping("/new")

public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee){

return new ResponseEntity<Employee>(

employeeService.saveEmployee(employee),

HttpStatus.CREATED

);

}

// get all employees

@GetMapping

public List<Employee> getAllEmployee(){

return employeeService.getAllEmployees();

}

// get employee by id

@GetMapping("{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") long employeeId){

return new ResponseEntity<Employee>(employeeService.getEmployeeById(employeeId),HttpStatus.OK);

}

}

1. Launch/Relaunch the application and test with Postman

A screenshot of a web page

Description automatically generated

**Update record by EmployeeID**

1. Modify the EmployeeService.java to add method that handles updating of record

package com.example.demo.service;

import java.util.List;

import com.example.demo.model.Employee;

public interface EmployeeService {

Employee saveEmployee(Employee employee);

List<Employee> getAllEmployees();

Employee getEmployeeById(long id);

**Employee updateEmployee(Employee employee, long id);**

}

1. Modify the EmployeeServiceImpl.java to add method that implements the new method in the service class

package com.example.demo.service.impl;

import java.util.List;

import java.util.Optional;

import org.springframework.stereotype.Service;

import com.example.demo.exception.ResourceNotFoundException;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

// dependency injection from the repository class

private EmployeeRepository employeeRepository;

// create constructor for this class

public EmployeeServiceImpl(EmployeeRepository employeeRepository) {

super();

this.employeeRepository = employeeRepository;

}

@Override

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

@Override

public List<Employee> getAllEmployees() {

// the code inside this method had been manually modified

return employeeRepository.findAll();

}

@Override

public Employee getEmployeeById(long id) {

// Optional<Employee> employee = employeeRepository.findById(id);

// if(employee.isPresent()) {

// return employee.get();

// }else {

// throw new ResourceNotFoundException("Employee", "Id", id);

// }

// or shorter by using lambda

return employeeRepository.findById(id).orElseThrow(

() -> new ResourceNotFoundException("Employee", "Id", id)

);

}

**@Override**

**public Employee updateEmployee(Employee employee, long id) {**

**// check if record with the id exists**

**Employee existingEmployee = employeeRepository.findById(id).orElseThrow(**

**() -> new ResourceNotFoundException("Employee", "Id", id)**

**);**

**existingEmployee.setFirstName(employee.getFirstName());**

**existingEmployee.setLastName(employee.getLastName());**

**existingEmployee.setEmail(employee.getEmail());**

**// save updated employee to database**

**employeeRepository.save(employee);**

**return existingEmployee;**

**}**

}

1. Modify the EmployeeController.java to add the RESTful API endpoint for the update record operation.

EmployeeController.java

package com.example.demo.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

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

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

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import com.example.demo.service.impl.EmployeeServiceImpl;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

super();

this.employeeService = employeeService;

}

@GetMapping("/test")

public String test() {

return "controller ok";

}

// API Endpoints for CRUD operations

// save new record

@PostMapping("/new")

public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee){

return new ResponseEntity<Employee>(

employeeService.saveEmployee(employee),

HttpStatus.CREATED

);

}

// get all employees

@GetMapping

public List<Employee> getAllEmployee(){

return employeeService.getAllEmployees();

}

// get employee by id

@GetMapping("{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") long employeeId){

return new ResponseEntity<Employee>(employeeService.getEmployeeById(employeeId),HttpStatus.OK);

}

// update record

@PutMapping("{id}")

public ResponseEntity<Employee> updateEmployee(

@PathVariable("id") long id,

@RequestBody Employee employee

)

{

return new ResponseEntity<Employee>(

employeeService.updateEmployee(employee,id),

HttpStatus.OK);

}

}

1. Launch/Relaunch application and test with Postman

A screenshot of a computer

Description automatically generated

**Delete record by EmployeeID**

1. Modify the EmployeeService.java to add method that will delete record by id.

EmployeeService.java

package com.example.demo.service;

import java.util.List;

import com.example.demo.model.Employee;

public interface EmployeeService {

Employee saveEmployee(Employee employee);

List<Employee> getAllEmployees();

Employee getEmployeeById(long id);

Employee updateEmployee(Employee employee, long id);

**void deleteEmployee(long id);**

}

1. Modify the EmployeeServiceImpl.java to implement method of the service class

EmployeeServiceImpl.java

package com.example.demo.service.impl;

import java.util.List;

import java.util.Optional;

import org.springframework.stereotype.Service;

import com.example.demo.exception.ResourceNotFoundException;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

// dependency injection from the repository class

private EmployeeRepository employeeRepository;

// create constructor for this class

public EmployeeServiceImpl(EmployeeRepository employeeRepository) {

super();

this.employeeRepository = employeeRepository;

}

@Override

public Employee saveEmployee(Employee employee) {

return employeeRepository.save(employee);

}

@Override

public List<Employee> getAllEmployees() {

// the code inside this method had been manually modified

return employeeRepository.findAll();

}

@Override

public Employee getEmployeeById(long id) {

// Optional<Employee> employee = employeeRepository.findById(id);

// if(employee.isPresent()) {

// return employee.get();

// } else {

// throw new ResourceNotFoundException("Employee", "Id", id);

// }

// or shorter by using lambda

return employeeRepository.findById(id).orElseThrow(

() -> new ResourceNotFoundException("Employee", "Id", id)

);

}

@Override

public Employee updateEmployee(Employee employee, long id) {

// check if record with the id exists

Employee existingEmployee = employeeRepository.findById(id).orElseThrow(

() -> new ResourceNotFoundException("Employee", "Id", id)

);

existingEmployee.setFirstName(employee.getFirstName());

existingEmployee.setLastName(employee.getLastName());

existingEmployee.setEmail(employee.getEmail());

// save updated employee to database

employeeRepository.save(employee);

return existingEmployee;

}

**@Override**

**public void deleteEmployee(long id) {**

**//check if record is existing before deleting**

**employeeRepository.findById(id).orElseThrow(**

**() -> new ResourceNotFoundException("Employee", "Id", id)**

**);**

**employeeRepository.deleteById(id);**

**}**

}

1. Modify the EmployeeController.java to add a RESTful API endpoint

package com.example.demo.controller;

import java.util.List;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

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

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

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

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

super();

this.employeeService = employeeService;

}

@GetMapping("/test")

// (GET) http://localhost:8080/api/employees/test

public String test() {

return "controller ok";

}

// API Endpoints for CRUD operations

// save new record

// (POST) http://localhost:8080/api/employees/new

@PostMapping("/new")

public ResponseEntity<Employee> saveEmployee(@RequestBody Employee employee){

return new ResponseEntity<Employee>(

employeeService.saveEmployee(employee),

HttpStatus.CREATED

);

}

// get all employees

// http://localhost:8080/api/employees

@GetMapping

public List<Employee> getAllEmployee(){

return employeeService.getAllEmployees();

}

// get employee by id

// (GET) http://localhost:8080/api/employees/1

@GetMapping("{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") long employeeId){

return new ResponseEntity<Employee>(employeeService.getEmployeeById(employeeId),HttpStatus.OK);

}

// update record

// (PUT) http://localhost:8080/api/employees/1

@PutMapping("{id}")

public ResponseEntity<Employee> updateEmployee(

@PathVariable("id") long id,

@RequestBody Employee employee

)

{

return new ResponseEntity<Employee>(

employeeService.updateEmployee(employee,id),

HttpStatus.OK

);

}

// delete record

// (DELETE) http://localhost:8080/api/employees/1

@DeleteMapping("{id}")

public ResponseEntity<String> updateEmployee(@PathVariable("id") long id)

{

employeeService.deleteEmployee(id);

return new ResponseEntity<String>(

"Employee Deleted Successfully",

HttpStatus.OK

);

}

}

1. Launch/Relaunch application and test with Postman

A screenshot of a computer

Description automatically generated

**Comprehend JUnit 5 platform and architecture and SoapUI features**

Notes:

* Junit is a Unit Testing Framework for Java
* Tests are written in Java
* Every time new code is added, the tests are re-executed.

What is Unit Testing?

* Testing of small chunks of code
* Helps early identification of defects.

Two types of Unit Testing

* Manual Testing
  + Testing manually without automation tool
  + Less reliable and time-consuming
* Automated Testing
  + Tested by tool support
  + More reliable and faster

**Demo 1:**

1. Create a new project in eclipse (new Java Project [non-maven]) “unittestdemo1”
2. Create a new class (ex. “Testclass1”) and a method (ex. “test1”) then use the @Test annotation to add Junit5 to your build path and add ‘requires junit;’ to module-info.java

A screen shot of a computer code

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer code

Description automatically generated

1. Write the test case

Testclass1.java

package unittestdemo1;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class JUnitTestclass1 {

@Test

public void test\_1() {

String correctstr = "I am the answer";

String tocheckstr = "I am answer";

assertEquals(correctstr, tocheckstr);

}

}

1. Add the test package to module-info.java

module unittestdemo1 {

requires junit;

**opens unittestdemo1;**

}

1. Run the test(s)

A screenshot of a computer

Description automatically generated

**Demo 2:**

1. Create a new maven/spring boot project (ex. “unittestdemo2”)
2. Create a method to test

DemoApplication.java

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

**public int addnums(int a, int b) {**

**return a+b;**

**}**

}

1. Create test method(s) in src/test/java

DemoApplicationTests.java

package com.example.demo;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class DemoApplicationTests {

@Test

void contextLoads() {

}

@Test

void testAddNums() {

DemoApplication app = new DemoApplication();

assertEquals(6, app.addnums(2, 2));

}

}

Note:

There are many different “assert” statements like assertSame, assertNotEquals, assertTrue, etc…

Demo 3

1. We will create a test for our API methods (not the endpoints yet)
2. You may want to add custom searching in your repository like so:

EmployeeRepository.java

package com.example.demo.repository;

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

import com.example.demo.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

**// built for unittest**

**Employee findByFirstName(String firstname);**

}

1. Here is our test case

RestcrudApplicationTests.java

package com.example.demo;

import static org.junit.jupiter.api.Assertions.assertNotNull;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

@SpringBootTest

class RestcrudApplicationTests {

@Autowired

EmployeeService es;

@Autowired

EmployeeRepository er;

@Test

void testAddEmployee() {

Employee newEmp = new Employee();

newEmp.setFirstName("efirstname");

newEmp.setLastName("eLastname");

newEmp.setDepartment("eDepartment");

newEmp.setEmail("eEmail");

es.saveEmployee(newEmp);

assertNotNull(er.findByFirstName(newEmp.getFirstName()));

}

}

1. Make sure the database engine (mysql) is running then try launching the test

Sample output (show not show any errors:

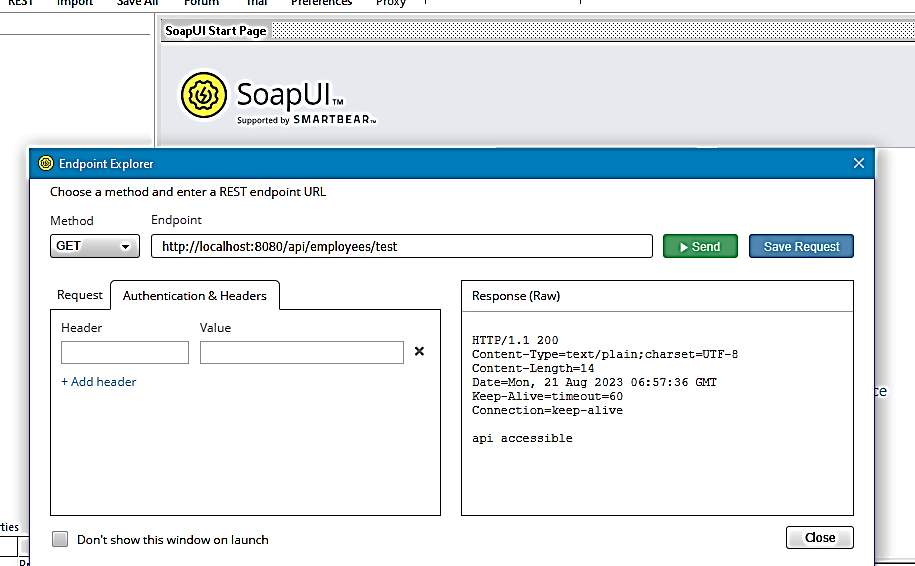
A screen shot of a computer

Description automatically generated

Load Testing using SOAP UI

SOAP UI can be used to test REST API endpoints

<https://www.soapui.org/downloads/soapui/>



A screenshot of a computer

Description automatically generated

Rclick on your api endpoint 🡪 generate test suite

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

You may set the ff:

* Threads = virtual users (ex. 100)
* Strategy = Load Test Strategy
* Test Delay = interval between tests in ms
* Random = randomness for delay
* Limit = in seconds / minutes