Calculate Sum of first N natural Numbers

void main() {

var num=3;

var temp=0;

for (int i=1;i<num+1;i++){

temp=temp+i;

}

print(temp);

}

Printing a Given Table

void main() {

var num=19;

var res;

for(int i=1;i<11;i++){

res=num\*i;

print("$num"+"\*"+"$i"+"="+"$res");

}

}

Calculate the factorial of given number

//5 factorial

//4\*3\*2\*1

void main() {

int n=5;

late int temp=1;

for (int i=1;i<=n;i++){

temp=i\*temp;

}

print(temp);

}

Factors of a Given Numver

//5 factorial

//4\*3\*2\*1

void main() {

int n=2;

late int temp=0;

for (int i=1;i<=n;i++){

if(n%i==0){

temp++;

}

}

print(temp);

}

Find the biggest of Three numbers

void main() {

var temp=0;

var a=10,b=40,c=13;

if(a>b&&a>c){

temp=a;

print(temp);

return;

}

else if(b>a&&b>c){

temp=b;

print(temp);

return;

}

temp =c;

print(temp);

}

To find out a leap year

main() {

var check=2004;

if(check%4==0){

print('yes it is a leap year'+check.toString());

return;

}

print('not la leap year');

}

Perfect Numbers

//sum of factors of number is equel to its number(excludes that number it self)

Ex:6

Factor of 6 is 1,2,3,6 so exclude 6 and sum of 1+2+3=6

void main() {

int n=33550336;

int temp=0;

int sum=0;

for (int i=1;i<=n;i++){

if(n%i==0){

sum=i+sum;

temp++;

}

}

print(temp);

if(n==sum-n){

print("perfectNummber");

return;

}else{

print("Not a perfect Number");

}

print(sum-n);

}

Merge 2 arrays alternately

void main() {

var list1=[2,86,9,67,86,9,06,];

var list2=[4,8,97,97,07,56,97];

var list3=[];

for(int i=0;i<list2.length;i++){

list3.add(list1[i]);

list3.add(list2[i]);

}

print(list3);

}

How to find Armstrong Number

import 'dart:math';

void main() {

//Arm strong number

// Armstrong number is sum of cubes of given number

int n=173;

List<String> x=n.toString().split('');

print(x);

int temp=0;

for(int i=0;i<x.length;i++){

print(x[i]);

temp= pow(int.parse(x[i]),3)+temp as int;

}

if(temp==n){

print("Number is armstrong");

}else{

print('Not Armstrong');

}

}

poindrome or not

//polindrome means a number is equel to riverse of number

//ex 121 is a polindrome

void main() {

int number=282;

var alist=number.toString().split('');

var rlist=alist.reversed.toList();

var temp='';

for(int i=0;i<alist.length;i++){

if(alist[i]!=rlist[i]){

temp=alist[i].toString()+temp;

}

}

if(temp==''){

print("Polindrome");

}else{

print("Not a Polindrome");

}

}

Printing Patterns

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

#include <stdio.h>

#include <stdlib.h>

main() {

for(int i=1;i<=5;i++){

for(int j=1;j<=5;j++){

//print(j);

printf("\*");

}

printf("\n");

}

}

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

#include <stdio.h>

#include <stdlib.h>

main() {

for(int i=1;i<=5;i++){

for(int j=1;j<=5;j++){

//print(j);

printf("\*");

}

printf("\n");

}

}

Printing below Pattern

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

#include <stdio.h>

int main()

{

int n=10;

for(int i=1;i<=n;i++){

for(int j=1;j<40-i;j++){

printf(" ");

}

for(int k=1;k<=2\*i-1;k++){

printf("\*");

}

printf("\n");

}

return 0;

}

Spring Auto Wiring

**package** com.rakesh.autowire;

**public** **class** Adress {

**private** String city;

**private** String street;

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** String getStreet() {

**return** street;

}

**public** **void** setStreet(String street) {

**this**.street = street;

}

**public** Adress(String city, String street) {

**super**();

**this**.city = city;

**this**.street = street;

}

@Override

**public** String toString() {

**return** "Adress [city=" + city + ", street=" + street + "]";

}

**public** Adress() {

**super**();

}

}

**package** com.rakesh.autowire;

**public** **class** Employee {

**private** Adress adress;

**public** Adress getAdress() {

**return** adress;

}

@Override

**public** String toString() {

**return** "Employee [adress=" + adress + "]";

}

**public** Employee() {

**super**();

}

**public** Employee(Adress adress) {

**super**();

**this**.adress = adress;

}

**public** **void** setAdress(Adress adress) {

**this**.adress = adress;

}

}

package com.rakesh.autowire;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Testing {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("com/rakesh/autowire/autoconfig.xml");

Employee emp= (Employee) context.getBean("emp");

System.out.println(emp);

}

}

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)*"*>

<bean class=*"com.rakesh.autowire.Adress"* name=*"adress"*>

<property name=*"city"*>

<value>Hyderabad</value>

</property>

<property name=*"street"*>

<value>Doyens township</value>

</property>

</bean>

<bean class=*"com.rakesh.autowire.Employee"* name=*"emp"* autowire=*"byName"*></bean>

</beans>

Constructor Injection

**package** com.rakesh.constructor;

**public** **class** Certificate {

**private** String name;

**public** Certificate(String name) {

**super**();

**this**.name = name;

}

@Override

**public** String toString() {

**return** "Certificate [name=" + name + "]";

}

}

**package** com.rakesh.constructor;

**public** **class** Person {

**private** String personId;

**private** String personName;

**private** Certificate certi;

**public** Person(String personId, String personName, Certificate certi) {

**super**();

**this**.personId = personId;

**this**.personName = personName;

**this**.certi = certi;

}

@Override

**public** String toString() {

**return** "Person [personId=" + personId + ", personName=" + personName + ", certi=" + certi + "]";

}

}

package com.rakesh.constructor;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Testci {

public static void main(String[] args) {

ApplicationContext ct = new ClassPathXmlApplicationContext("com/rakesh/constructor/ciconfigxml.xml");

Person p1 = (Person) ct.getBean("person");

System.out.println(p1);

}

}

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)*"*>

<!-- A simple bean definition -->

<bean class=*"com.rakesh.constructor.Person"* name=*"person"*>

<constructor-arg value=*"12"* />

<constructor-arg value=*"raju"* />

<constructor-arg ref=*"certi"* />

</bean>

<bean class=*"com.rakesh.constructor.Certificate"* name=*"certi"*>

<constructor-arg value=*"certificateName"* />

</bean>

</beans>

Injecting Reference Type Object

**package** com.rakesh.ref;

**public** **class** A {

**private** String name;

**private** B b;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** B getB() {

**return** b;

}

**public** A(String name, B b) {

**super**();

**this**.name = name;

**this**.b = b;

}

**public** A() {

**super**();

}

@Override

**public** String toString() {

**return** "A [name=" + name + ", b=" + b + "]";

}

**public** **void** setB(B b) {

**this**.b = b;

}

}

**package** com.rakesh.ref;

**public** **class** B {

**public** B() {

**super**();

}

**private** String userid;

**public** String getUserid() {

**return** userid;

}

**public** **void** setUserid(String userid) {

**this**.userid = userid;

}

@Override

**public** String toString() {

**return** "B [userid=" + userid + "]";

}

**public** B(String userid) {

**super**();

**this**.userid = userid;

}

}

package com.rakesh.ref;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

public static void main(String[] args) {

ApplicationContext ct = new ClassPathXmlApplicationContext("com/rakesh/ref/refconfig.xml");

A a = (A) ct.getBean("aref");

System.out.println(a.getB().getUserid());

}

}

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

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)*"*>

<!-- A simple bean definition -->

<bean class=*"com.rakesh.ref.A"* name=*"aref"*>

<property name=*"name"* value=*"rakesh"*>

</property>

<property name=*"b"*>

<ref bean=*"bref"*/>

</property>

</bean>

<bean class =*"com.rakesh.ref.B"* name=*"bref"*>

<property name=*"userid"*>

<value>165</value>

</property>

</bean>

</beans>

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

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

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*](http://www.springframework.org/schema/beans/spring-beans-3.0.xsd)*"*>

<!-- A simple bean definition -->

<bean class=*"com.springcore.sterotype.Student"* name=*"aref"*>

</bean>

<bean class=*"com.springcore.sterotype.B"*>

</bean>

</beans>

Spring Jdbc

Config.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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.2.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context-4.2.xsd*](http://www.springframework.org/schema/context/spring-context-4.2.xsd)*"*>

<!-- data source bean -->

<bean

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*

name=*"dataSource"*>

<property name=*"driverClassName"*

value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/rakeshdb"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"rakesh12@"*></property>

</bean>

<!-- jdbc template bean -->

<bean class=*"org.springframework.jdbc.core.JdbcTemplate"*

name=*"jdbc"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<!--implemented class bean -->

<bean class=*"com.rakesh.springjdbc.dao.StudentDaoImpl"*

name=*"studentd"*>

<property name=*"template"* ref=*"jdbc"*></property>

</bean>

</beans>

StudentDao:

**package** com.rakesh.springjdbc.dao;

**import** java.util.List;

**import** com.rakesh.springjdbc.entities.Student;

**public** **interface** StudentDao {

**public** **int** insertStudent(Student student);

**public** **int** changeStudent(Student student);

**public** **int** deleteStudent(Student student);

**public** Student getStudent(Student student);

**public** List<Student> getAllStudents();

}

StudentDaoImpl

**package** com.rakesh.springjdbc.dao;

**import** java.util.List;

**import** java.util.Map;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.core.RowMapper;

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

**import** com.rakesh.springjdbc.entities.Student;

**public** **class** StudentDaoImpl **implements** StudentDao {

JdbcTemplate template;

**public** JdbcTemplate getTemplate() {

**return** template;

}

**public** **void** setTemplate(JdbcTemplate template) {

**this**.template = template;

}

**public** **int** insertStudent(Student s) {

String query = "insert into student\_table(id,name,city) values(?,?,?)";

**int** reslut = template.update(query, s.getId(), s.getName(), s.getCity());

**return** reslut;

}

**public** **int** changeStudent(Student student) {

String uQuery = " UPDATE student\_table SET name = ?, city=? WHERE id = ?";

**int** result = template.update(uQuery, student.getName(), student.getCity(), student.getId());

**return** result;

}

**public** **int** deleteStudent(Student student) {

String query = "delete from student\_table where id =?";

**int** res = template.update(query, student.getId());

**return** res;

}

**public** Student getStudent(Student student) {

String query = "select \* from student\_table where id =?";

RowMapper<Student> rm = **new** RowMapperImpl();

Student s1 = template.queryForObject(query, rm, student.getId());

**return** s1;

}

**public** List<Student> getAllStudents() {

RowMapper<Student> rm1 = **new** RowMapperImpl();

String selectAllQuery = "Select \* from student\_table";

List<Student> students = template.query(selectAllQuery, rm1);

**return** students;

}

}

RowMapper class

**package** com.rakesh.springjdbc.dao;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper;

**import** com.rakesh.springjdbc.entities.Student;

**public** **class** RowMapperImpl **implements** RowMapper<Student> {

**public** Student mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

Student s1 = **new** Student();

s1.setId(rs.getInt(1));

s1.setCity(rs.getString(3));

s1.setName(rs.getString(2));

**return** s1;

}

}

Mainclass

**package** com.rakesh.springjdbc;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** com.rakesh.springjdbc.dao.StudentDao;

**import** com.rakesh.springjdbc.entities.Student;

/\*\*

\* Hello world!

\*

\*/

**public** **class** App {

**public** **static** **void** main(String[] args) {

System.***out***.println("Hello World!");

ApplicationContext context = **new** ClassPathXmlApplicationContext("com/rakesh/springjdbc/config.xml");

StudentDao st = (StudentDao) context.getBean("studentd");

List<Student> s2 = st.getAllStudents();

System.***out***.println(s2);

}

}

Student class

**package** com.rakesh.springjdbc.entities;

**public** **class** Student {

**private** **int** id;

**private** String name;

**private** String city;

**public** Student() {

}

**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;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + ", city=" + city + "]";

}

**public** Student(**int** id, String name, String city) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.city = city;

}

}

Java configuration with Spring Jdbc

**package** com.rakesh.springjdbc;

**import** javax.sql.DataSource;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;

**import** org.springframework.jdbc.datasource.DriverManagerDataSource;

**import** com.rakesh.springjdbc.dao.StudentDao;

**import** com.rakesh.springjdbc.dao.StudentDaoImpl;

@Configuration

**public** **class** JdbcConfig {

@Bean("dataSource")

**public** DataSource getDataSource() {

DriverManagerDataSource driverManagerDataSource = **new** DriverManagerDataSource();

driverManagerDataSource.setUrl("jdbc:mysql://localhost:3306/rakeshdb");

driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");

driverManagerDataSource.setUsername("root");

driverManagerDataSource.setPassword("rakesh12@");

**return** driverManagerDataSource;

}

@Bean("jdbc")

**public** JdbcTemplate getJdbcTemplate() {

JdbcTemplate template = **new** JdbcTemplate();

template.setDataSource(getDataSource());

**return** template;

}

@Bean("studentd")

**public** StudentDao getStudent() {

StudentDaoImpl impl = **new** StudentDaoImpl();

impl.setTemplate(getJdbcTemplate());

**return** impl;

}

}

**package** com.rakesh.springjdbc;

**import** java.util.List;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** com.rakesh.springjdbc.dao.StudentDao;

**import** com.rakesh.springjdbc.entities.Student;

/\*\*

\* Hello world!

\*

\*/

**public** **class** App {

**public** **static** **void** main(String[] args) {

System.***out***.println("Hello World!");

ApplicationContext context = **new** AnnotationConfigApplicationContext(JdbcConfig.**class**);

StudentDao st = (StudentDao) context.getBean("studentd");

List<Student> s2 = st.getAllStudents();

System.***out***.println(s2);

}

}

With AutoWired Annotation

**package** com.rakesh.springjdbc;

**import** javax.sql.DataSource;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.datasource.DataSourceTransactionManager;

**import** org.springframework.jdbc.datasource.DriverManagerDataSource;

**import** com.rakesh.springjdbc.dao.StudentDao;

**import** com.rakesh.springjdbc.dao.StudentDaoImpl;

@Configuration

@ComponentScan({"com.rakesh.springjdbc"})

**public** **class** JdbcConfig {

@Bean("dataSource")

**public** DataSource getDataSource() {

DriverManagerDataSource driverManagerDataSource = **new** DriverManagerDataSource();

driverManagerDataSource.setUrl("jdbc:mysql://localhost:3306/rakeshdb");

driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");

driverManagerDataSource.setUsername("root");

driverManagerDataSource.setPassword("rakesh12@");

**return** driverManagerDataSource;

}

@Bean("jdbc")

**public** JdbcTemplate getJdbcTemplate() {

JdbcTemplate template = **new** JdbcTemplate();

template.setDataSource(getDataSource());

**return** template;

}

// @Bean("studentd")

// public StudentDao getStudent() {

// StudentDaoImpl impl = new StudentDaoImpl();

// impl.setTemplate(getJdbcTemplate());

// return impl;

// }

}

package com.rakesh.springjdbc.dao;

import java.util.List;

import java.util.Map;

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

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Component;

import com.rakesh.springjdbc.entities.Student;

@Component("studentd")

public class StudentDaoImpl implements StudentDao {

@Autowired

JdbcTemplate template;

public JdbcTemplate getTemplate() {

return template;

}

public void setTemplate(JdbcTemplate template) {

this.template = template;

}

public int insertStudent(Student s) {

String query = "insert into student\_table(id,name,city) values(?,?,?)";

int reslut = template.update(query, s.getId(), s.getName(), s.getCity());

return reslut;

}

public int changeStudent(Student student) {

String uQuery = " UPDATE student\_table SET name = ?, city=? WHERE id = ?";

int result = template.update(uQuery, student.getName(), student.getCity(), student.getId());

return result;

}

public int deleteStudent(Student student) {

String query = "delete from student\_table where id =?";

int res = template.update(query, student.getId());

return res;

}

public Student getStudent(Student student) {

String query = "select \* from student\_table where id =?";

RowMapper<Student> rm = new RowMapperImpl();

Student s1 = template.queryForObject(query, rm, student.getId());

return s1;

}

public List<Student> getAllStudents() {

RowMapper<Student> rm1 = new RowMapperImpl();

String selectAllQuery = "Select \* from student\_table";

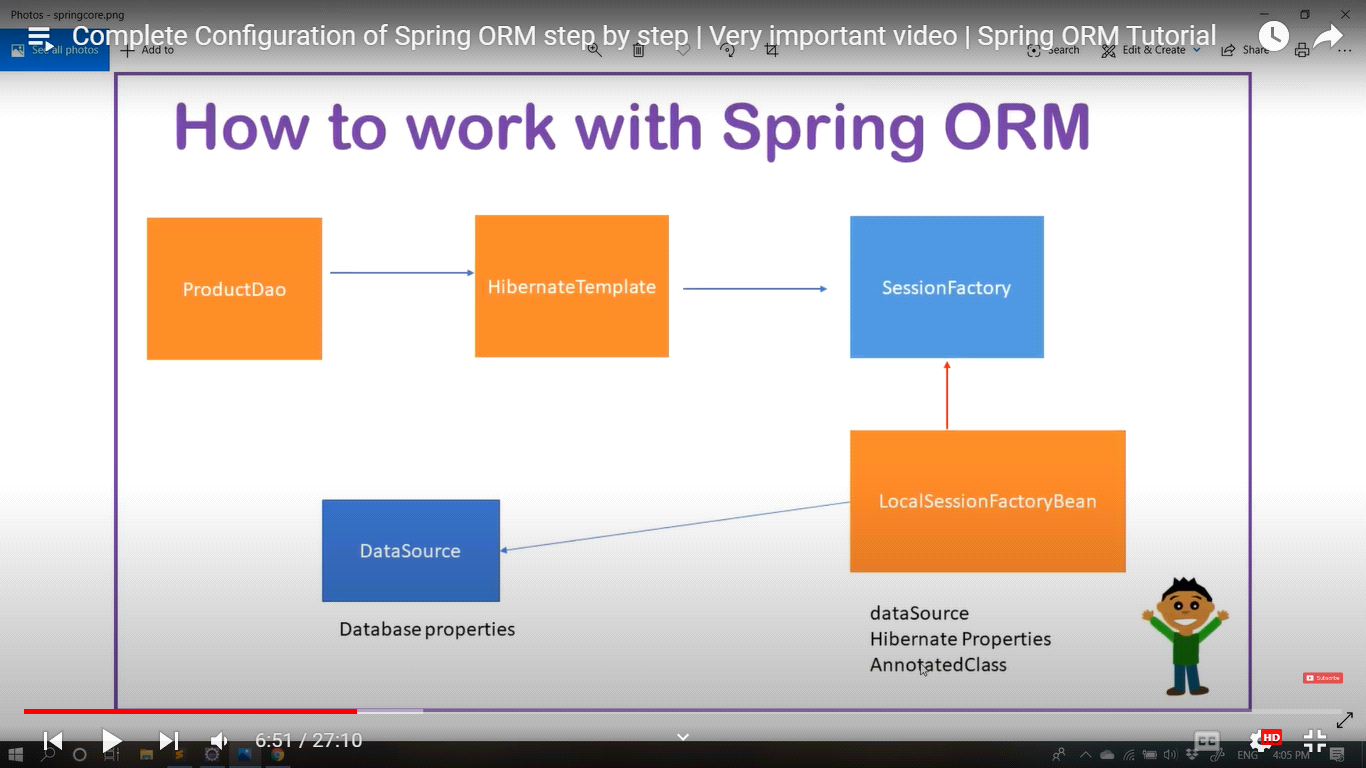
List<Student> students = template.query(selectAllQuery, rm1);

return students;

}

}

Spring Hibernate



Porm.xml

<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* [*http://maven.apache.org/xsd/maven-4.0.0.xsd*](http://maven.apache.org/xsd/maven-4.0.0.xsd)*"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.rakesh</groupId>

<artifactId>springorm</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>springorm</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.2.3.RELEASE</version>

</dependency>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.3.RELEASE</version>

</dependency>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.2.3.RELEASE</version>

</dependency>

<!-- <https://mvnrepository.com/artifact/com.mysql/mysql-connector-j> -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.31</version>

</dependency>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>5.2.3.RELEASE</version>

</dependency>

<!-- <https://mvnrepository.com/artifact/org.hibernate/hibernate-core> -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.4.29.Final</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

Student entity:

**package** com.rakesh.springorm.entities;

**import** javax.persistence.Column;

**import** javax.persistence.Table;

**import** org.hibernate.annotations.~~Entity~~;

@~~Entity~~

@Table(name = "student\_table")

**public** **class** Student {

@Column(name = "id")

**private** **int** id;

@Column(name = "name")

**private** String name;

@Column(name = "city")

**private** String city;

**public** Student(**int** id, String name, String city) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.city = city;

}

**public** Student() {

}

**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;

}

**public** String getCity() {

**return** city;

}

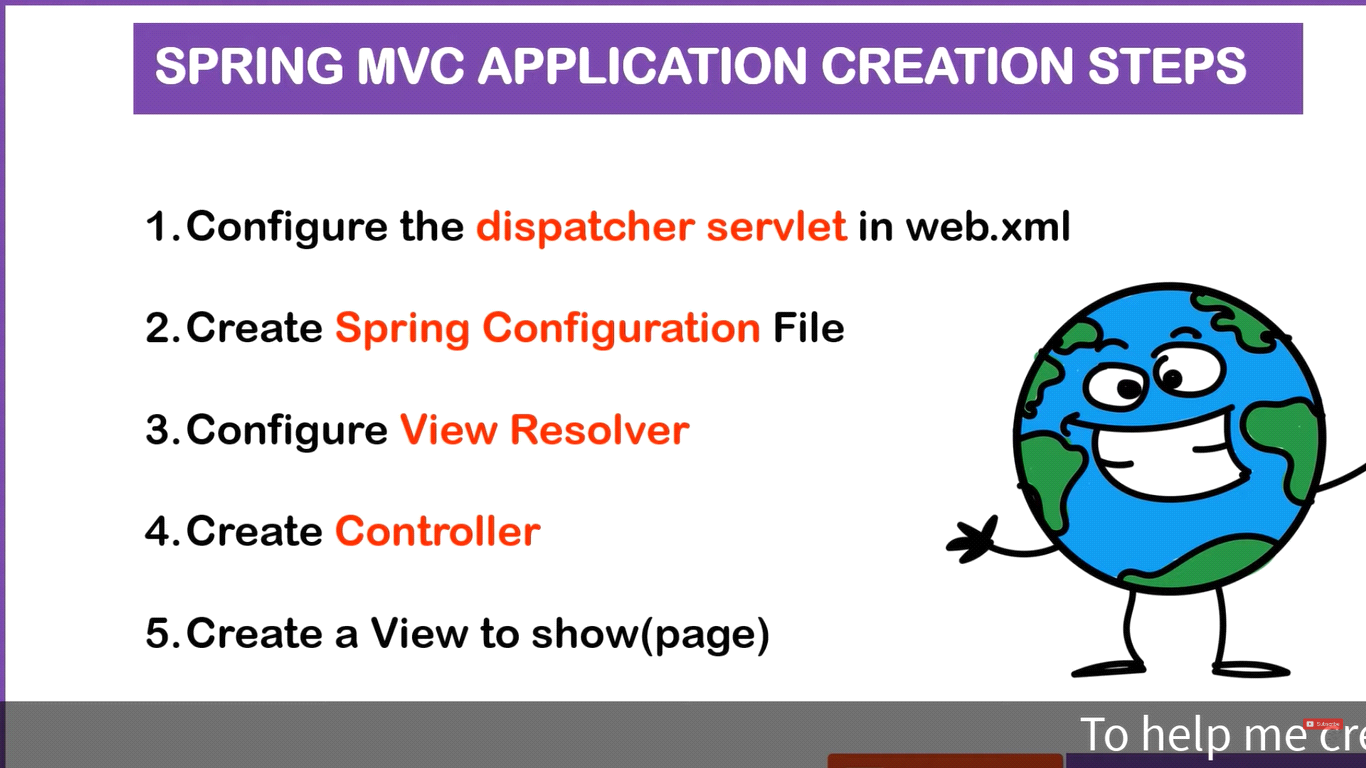
**public** **void** setCity(String city) {

**this**.city = city;

}

}

Spring Mvc



Step1: in web.xml file

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"<http://java.sun.com/dtd/web-app_2_3.dtd>" >

<web-app>

<display-name>Archetype Created Web Application</display-name>

<!-- Configure Dispatcher Servlet -->

<servlet>

<servlet-name>Spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

Step2&3:spring configuration&view resolver with bean

<?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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.2.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context-4.2.xsd*](http://www.springframework.org/schema/context/spring-context-4.2.xsd)

[*http://www.springframework.org/schema/tx*](http://www.springframework.org/schema/tx)

[*http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*](http://www.springframework.org/schema/tx/spring-tx-4.2.xsd)

*"*>

<context:component-scan base-package=*"springmvc.controller"*></context:component-scan>

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*

name=*"viewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

</beans>

Step4: creating controller

**package** springmvc.controller;

**import** org.springframework.stereotype.Controller;

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

@Controller

**public** **class** HomeController {

@RequestMapping("/home")

**public** String home() {

System.***out***.println("this is home url");

**return** "index";

}

}

Step5: Adding a view

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

hello

</body>

</html>

Sending Data from Controller To View

Controller:

@Controller

**public** **class** HomeController {

@RequestMapping("/home")

**public** String home(Model model) {

model.addAttribute("name", "rakesh only");

List<Integer> list = **new** ArrayList<Integer>();

list.add(1);

list.add(2);

list.add(3);

model.addAttribute("list",list);

System.***out***.println("this is home url");

**return** "index";

}

@RequestMapping("/about")

**public** String aboutUs() {

System.***out***.println("this is home url");

**return** "about";

}

}

View:

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@page isELIgnored=*"false"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

hello

<%

//String name = (String) request.getAttribute("name");

%>

${name}

<h1>${list}</h1>

</body>

</html>

Dependency:

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.1.2</version>

</dependency>

Sending data from view to Controller

With @RequestParam:

<!doctype html>

<html lang=*"en"*>

<head>

<!-- Required meta tags -->

<meta charset=*"utf-8"*>

<meta name=*"viewport"*

content=*"width=device-width, initial-scale=1, shrink-to-fit=no"*>

<!-- Bootstrap CSS -->

<link rel=*"stylesheet"*

href=*"https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"*

integrity=*"sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"*

crossorigin=*"anonymous"*>

<title>Hello, world!</title>

</head>

<body>

<div class=*"container mt-5"*>

<h3 class=*"text-center"*>Registration Form</h3>

<form action=*"processForm"* method=*"post"*>

<div class=*"form-group "*>

<label for=*"exampleInputEmail1"*>Email address</label> <input

type=*"email"* class=*"form-control"* id=*"exampleInputEmail1"*

name=*"email"* aria-describedby=*"emailHelp"* placeholder=*"Enter email"*>

</div>

<div class=*"form-group "*>

<label for=*"exampleInputEmail1"*>User Name</label> <input type=*"text"*

name=*"userName"* class=*"form-control"* id=*"userName"*

aria-describedby=*"emailHelp"* placeholder=*"Enter UserName"*>

</div>

<div class=*"form-group "*>

<label for=*"exampleInputEmail1"*>Password</label> <input

type=*"password"* class=*"form-control"* id=*"UserName"* name=*"pass"*

aria-describedby=*"emailHelp"* placeholder=*"Enter Password"*>

</div>

<div class=*"text-center"*>

<input type=*"submit"*>

</div>

</form>

</div>

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src=*"https://code.jquery.com/jquery-3.2.1.slim.min.js"*

integrity=*"sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"*

crossorigin=*"anonymous"*></script>

<script

src=*"https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"*

integrity=*"sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"*

crossorigin=*"anonymous"*></script>

<script

src=*"https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"*

integrity=*"sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"*

crossorigin=*"anonymous"*></script>

</body>

</html>

Controller:

**package** springmvc.controller;

**import** javax.servlet.annotation.MultipartConfig;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

@Controller

**public** **class** ContactController {

@RequestMapping("/contact")

**public** String showForm() {

**return** "contact";

}

@RequestMapping(path = "/processForm", method = RequestMethod.***POST***)

**public** String contactUser(@RequestParam("userName") String name, @RequestParam("email") String email,

@RequestParam("pass") String password, Model model) {

model.addAttribute("userName", name);

model.addAttribute("email", email);

model.addAttribute("pass", password);

**return** "success";

}

}

Success.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@page isELIgnored=*"false"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h1>${userName}</h1>

<h1>${email}</h1>

</body>

</html>

Using @ModelAttribute

**package** springmvc.controller;

**import** javax.servlet.annotation.MultipartConfig;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

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

**import** org.springframework.web.bind.annotation.RequestMethod;

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

**import** springmvc.model.User;

@Controller

**public** **class** ContactController {

@RequestMapping("/contact")

**public** String showForm() {

**return** "contact";

}

@RequestMapping(path = "/processForm", method = RequestMethod.***POST***)

**public** String contactUser(@ModelAttribute User user) {

**return** "success";

}

}

Writing common method with @Model Attribute

@ModelAttribute

**public** **void** commonMethod(Model m) {

m.addAttribute("Header", "Learn Code with Rakesh");

m.addAttribute("description", "You can Learn Code with Rakesh kulla");

}

Hibernate configuration

(not working)

Spring-servlet.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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.2.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context-4.2.xsd*](http://www.springframework.org/schema/context/spring-context-4.2.xsd)

[*http://www.springframework.org/schema/tx*](http://www.springframework.org/schema/tx)

[*http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*](http://www.springframework.org/schema/tx/spring-tx-4.2.xsd)

*"*>

<context:component-scan base-package=*"springmvc"*></context:component-scan>

<!-- hibername class bean -->

<bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*

name=*"hibernateTemplate"*>

<property name=*"sessionFactory"* ref=*"factory"*></property>

</bean>

<!-- session factory class bean -->

<bean

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*

name=*"factory"*>

<property name=*"dataSource"* ref=*"ds"*></property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQL57Dialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

</props>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>

springmvc.model.User

</value>

</list>

</property>

</bean>

<!-- DataSource class -->

<bean

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"diverClassName"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/rakeshdb"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"rakesh12@"*></property>

</bean>

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*

name=*"viewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

</beans>

contactControoler.java

package springmvc.controller;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

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

import springmvc.model.User;

import springmvc.service.UserService;

@Controller

public class ContactController {

@Autowired

private UserService userService;

@ModelAttribute

public void commonMethod(Model m) {

m.addAttribute("Header", "Learn Code with Rakesh");

m.addAttribute("description", "You can Learn Code with Rakesh kulla");

}

@RequestMapping("/contact")

public String showForm() {

return "contact";

}

@RequestMapping(path = "/processForm", method = RequestMethod.POST)

public String contactUser(@ModelAttribute User user) {

this.userService.createUser(user);

return "success";

}

}

Userdao.java

package springmvc.dao;

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

import org.springframework.orm.hibernate5.HibernateTemplate;

import org.springframework.stereotype.Repository;

import springmvc.model.User;

@Repository

public class UserDao {

@Autowired

private HibernateTemplate hibernateTemplate;

public int saveUser(User user) {

Integer id = (Integer) this.hibernateTemplate.save(user);

return id;

}

}

User.java

package springmvc.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;;

@Entity

public class User {

@Id

private int id;

@GeneratedValue(strategy = GenerationType.AUTO)

private String userName;

private String email;

private String password;

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

@Override

public String toString() {

return "User [userName=" + userName + ", email=" + email + ", password=" + password + "]";

}

public User(String userName, String email, String password) {

super();

this.userName = userName;

this.email = email;

this.password = password;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

UserService.java

package springmvc.service;

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

import org.springframework.stereotype.Service;

import springmvc.dao.UserDao;

import springmvc.model.User;

@Service

public class UserService {

@Autowired

UserDao dao;

public int createUser(User user) {

return this.dao.saveUser(user);

}

}

Web.xml

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"<http://java.sun.com/dtd/web-app_2_3.dtd>" >

<web-app>

<display-name>Archetype Created Web Application</display-name>

<!-- Configure Dispatcher Servlet -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

Porm.xml

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

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0* [*http://maven.apache.org/xsd/maven-4.0.0.xsd*](http://maven.apache.org/xsd/maven-4.0.0.xsd)*"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>

<modelVersion>4.0.0</modelVersion>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.2.3.RELEASE</version>

<name>Spring Web MVC</name>

<description>Spring Web MVC</description>

<url>https://github.com/spring-projects/spring-framework</url>

<organization>

<name>Spring IO</name>

<url>https://spring.io/projects/spring-framework</url>

</organization>

<licenses>

<license>

<name>Apache License, Version 2.0</name>

<url>https://www.apache.org/licenses/LICENSE-2.0</url>

<distribution>repo</distribution>

</license>

</licenses>

<developers>

<developer>

<id>jhoeller</id>

<name>Juergen Hoeller</name>

<email>jhoeller@pivotal.io</email>

</developer>

</developers>

<scm>

<connection>scm:git:git://github.com/spring-projects/spring-framework</connection>

<developerConnection>scm:git:git://github.com/spring-projects/spring-framework</developerConnection>

<url>https://github.com/spring-projects/spring-framework</url>

</scm>

<issueManagement>

<system>GitHub</system>

<url>https://github.com/spring-projects/spring-framework/issues</url>

</issueManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-expression</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.2.3.RELEASE</version>

<scope>compile</scope>

</dependency>

</dependencies>

</project>

package com.rakesh.springorm;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.rakesh.springorm.dao.StudentDao;

import com.rakesh.springorm.entities.Student;

/\*\*

\* Hello world!

\*

\*/

public class App {

public static void main(String[] args) {

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

StudentDao dao = (StudentDao) context.getBean("studentd");

Student st1 = new Student();

st1.setId(1);

st1.setName("ravi");

st1.setCity("warangal");

dao.insert(st1);

}

}

package com.rakesh.springorm.dao;

import javax.transaction.Transactional;

import org.springframework.orm.hibernate5.HibernateTemplate;

import com.rakesh.springorm.entities.Student;

public class StudentDao {

private HibernateTemplate hibernateTemplate;

@Transactional

public int insert(Student student) {

Integer i = (Integer) this.hibernateTemplate.save(student);

return i;

}

public HibernateTemplate getHibernateTemplate() {

return hibernateTemplate;

}

public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {

this.hibernateTemplate = hibernateTemplate;

}

}

**package** com.rakesh.springorm.entities;

**import** javax.persistence.Column;

**import** javax.persistence.Table;

**import** javax.persistence.Entity;

@Entity

**public** **class** Student {

**private** **int** id;

@Column(name = "name")

**private** String name;

@Column(name = "city")

**private** String city;

**public** Student(**int** id, String name, String city) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.city = city;

}

**public** Student() {

}

**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;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

<?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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.2.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context-4.2.xsd*](http://www.springframework.org/schema/context/spring-context-4.2.xsd)

[*http://www.springframework.org/schema/tx*](http://www.springframework.org/schema/tx)

[*http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*](http://www.springframework.org/schema/tx/spring-tx-4.2.xsd)

*"*>

<tx:annotation-driven/>

<!-- student dao bean -->

<bean class=*"com.rakesh.springorm.dao.StudentDao"*

name=*"studentd"*>

<property name=*"hibernateTemplate"* ref=*"hiber"*></property>

</bean>

<!-- hibernate bean -->

<bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*

name=*"hiber"*>

<property name=*"sessionFactory"* ref=*"localBean"*></property>

</bean>

<!-- localSessionFactoryBean -->

<bean

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*

name=*"localBean"*>

<property name=*"dataSource"* ref=*"ds"*></property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQL57Dialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

</props>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>

com.rakesh.springorm.entities.Student

</value>

</list>

</property>

</bean>

<!-- data source bean -->

<bean

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*

name=*"ds"*>

<property name=*"driverClassName"*

value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/rakeshdb"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"rakesh12@"*></property>

</bean>

<bean

class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*

name=*"transactionManager"*>

<property name=*"sessionFactory"* ref=*"localBean"*></property>

</bean>

</beans>

<?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"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

[*http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-4.2.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context-4.2.xsd*](http://www.springframework.org/schema/context/spring-context-4.2.xsd)

[*http://www.springframework.org/schema/tx*](http://www.springframework.org/schema/tx)

[*http://www.springframework.org/schema/tx/spring-tx-4.2.xsd*](http://www.springframework.org/schema/tx/spring-tx-4.2.xsd)

*"*>

<context:component-scan base-package=*"springmvc"*></context:component-scan>

<tx:annotation-driven />

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*

name=*"viewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

<!-- hibernate Bean -->

<bean class=*"org.springframework.orm.hibernate5.HibernateTemplate"*

name=*"hibernateref"*>

<property name=*"sessionFactory"* ref=*"factory"*></property>

<property name=*"checkWriteOperations"* value=*"false"*></property>

</bean>

<!-- local Session Factory Bean -->

<bean

class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*

name=*"factory"*>

<property name=*"dataSource"* ref=*"ds"*></property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQL57Dialect</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop>

</props>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>

springmvc.model.User

</value>

</list>

</property>

</bean>

<!-- DataSource class -->

<bean

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"* name=*"ds"*>

<property name=*"driverClassName"*

value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306/rakeshdb"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"rakesh12@"*></property>

</bean>

<!-- transatcion Manager Bean -->

<bean

class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*

name=*"transactionManager"*>

<property name=*"sessionFactory"* ref=*"localBean"*></property>

</bean>

<!-- contact controller bean -->

<bean class=*"springmvc.controller.ContactController"*

name=*"controller"*>

<property name=*"userService"* ref=*"userDao"*></property>

</bean>

<!-- userDao bean -->

<bean class=*"springmvc.dao.UserDao"* name=*"userDao"*>

<property name=*"hibernateTemplate"* ref=*"hibernateTemplate"*></property>

</bean>

<bean class=*"springmvc.service.UserService"* name=*"userService"*>

<property name=*"dao"* ref=*"userDao"*></property>

</bean>

</beans>

How to change Java jdk version in Spring tool suite or eclipse

step1:right click on your project in sts

step2:select build path

step3: select configure build path

step4:select libraries

step5:expand madule path and tap on jre system library

step6:click on edit

step7:click on alternate jre radio button

step8:click on selected jre's

step9:then click on Add

step10:then select standerd vm

step 11:then select Next

step12:then select Directory and point your jdk location

step13:then finish and tick the your new jdk and apply and close

Spring Boot Data Jpa

Step1:Create a spring boot starter Project in sts or spring initializer.io

Step2:First connect Application to Database to do this

we need create a Database in Sql workbench,then configure database in application.properties

spring.datasource.name=test

spring.datasource.url=jdbc:mysql://localhost:3306/jpadtest?useSSL=false

spring.datasource.username=root

spring.datasource.password=rakesh12

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

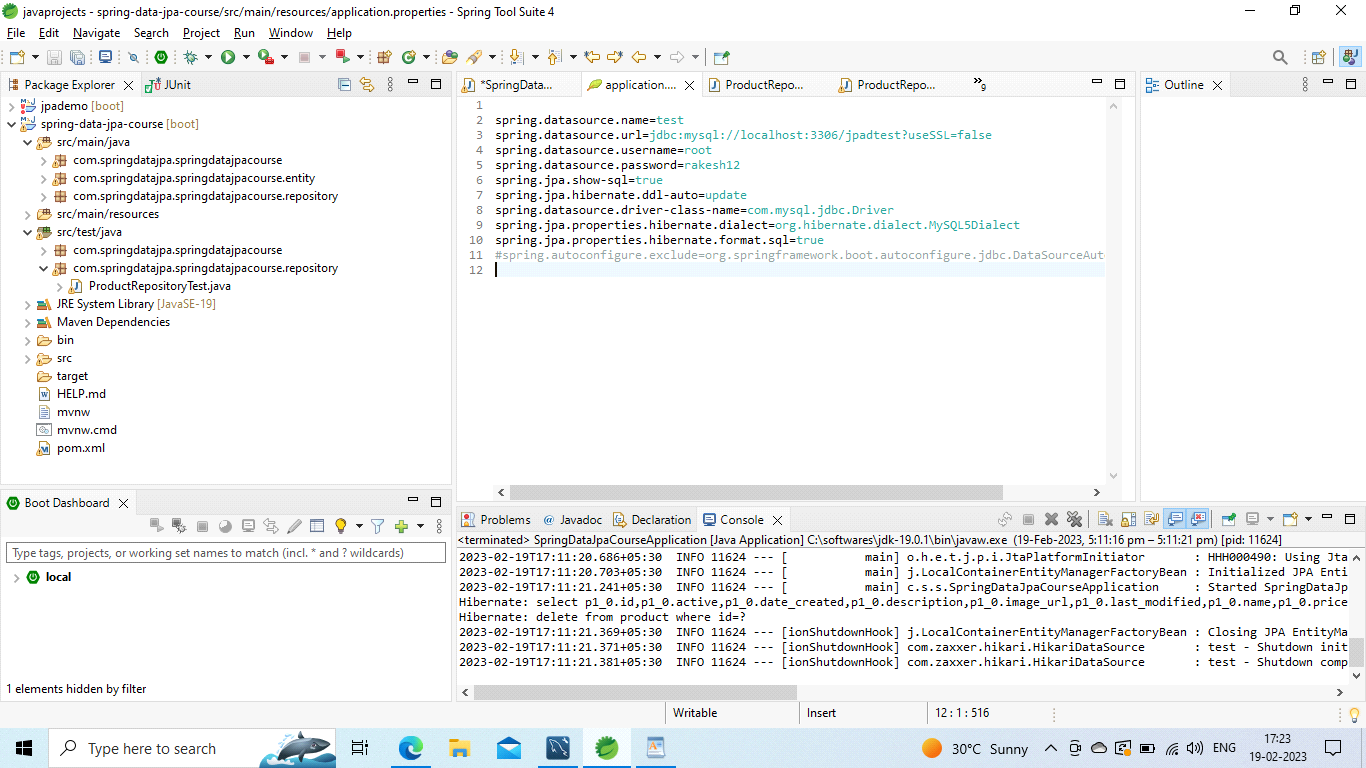
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

spring.jpa.properties.hibernate.format.sql=true

Step3: Then Run the code Make sure that no errors in console

Step4:Create two Folders exactly descendent to the directory where your main.java file exists.



take reference for above image

in this two directories One for Entity(Model) and Other for Repository

Step 5: then create entity class

package com.springdatajpa.springdatajpacourse.entity;

import java.math.BigDecimal;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

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 jakarta.persistence.UniqueConstraint;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

@Entity

//@Table(name = "products", schema = "jpadtest", uniqueConstraints = {

// @UniqueConstraint(name = "spu\_unique", columnNames = "stock\_skeeping\_unit") })

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Column(name = "stock\_keeping\_unit")

private String sku;

@Column(nullable = false)

private String name;

private String description;

private BigDecimal price;

private boolean active;

private String imageUrl;

@CreationTimestamp

private LocalDateTime dateCreated;

@UpdateTimestamp

private LocalDateTime lastModified;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getSku() {

return sku;

}

public void setSku(String sku) {

this.sku = sku;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public BigDecimal getPrice() {

return price;

}

public void setPrice(BigDecimal price) {

this.price = price;

}

public boolean isActive() {

return active;

}

public void setActive(boolean active) {

this.active = active;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public LocalDateTime getDateCreated() {

return dateCreated;

}

public void setDateCreated(LocalDateTime dateCreated) {

this.dateCreated = dateCreated;

}

public LocalDateTime getLastModified() {

return lastModified;

}

public void setLastModified(LocalDateTime lastModified) {

this.lastModified = lastModified;

}

public Product(Long id, String sku, String name, String description, BigDecimal price, boolean active,

String imageUrl, LocalDateTime dateCreated, LocalDateTime lastModified) {

super();

this.id = id;

this.sku = sku;

this.name = name;

this.description = description;

this.price = price;

this.active = active;

this.imageUrl = imageUrl;

this.dateCreated = dateCreated;

this.lastModified = lastModified;

}

public Product() {

super();

// TODO Auto-generated constructor stub

}

@Override

public String toString() {

return "Product [id=" + id + ", sku=" + sku + ", name=" + name + ", description=" + description + ", price="

+ price + ", active=" + active + ", imageUrl=" + imageUrl + ", dateCreated=" + dateCreated

+ ", lastModified=" + lastModified + "]";

}

}

In this Entity Iam try to use Lobok dependency to Avoid Getters and Getters but its not Working As Expected

Step 6: then we need to Create a Repository interface and extend it with JpaRepository Interface

**package** com.springdatajpa.springdatajpacourse.repository;

**import** java.time.LocalDateTime;

**import** java.util.List;

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

**import** com.springdatajpa.springdatajpacourse.entity.Product;

**public** **interface** ProductRepository **extends** JpaRepository<Product, Long> {

}

Step7: Then in main class we can directly Access the repository

package com.springdatajpa.springdatajpacourse;

import java.math.BigDecimal;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

import org.springframework.context.ApplicationContext;

import com.springdatajpa.springdatajpacourse.entity.Product;

import com.springdatajpa.springdatajpacourse.repository.ProductRepository;

@SpringBootApplication()

public class SpringDataJpaCourseApplication {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(SpringDataJpaCourseApplication.class, args);

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

repository.deleteById(403L);

}

}

But In our Case Iam Used Junit Test Cases to Test Piece of Code

Step1: For that we need to Create a Directory inside src/test/java

same as exactly descendent of directory where main.java exist

Step2:Create a Class like ProductRepositoryTest

Step3:and then Annotate this class with @SpingBootTest

Step4:then Create a reference for Product Repository

@Autowired

**private** ProductRepository repository;

step5: you can wrire all your test cases by annotating with @Test

@Test

**void** saveMethod() {

Product product = **new** Product();

product.setName("product2");

product.setDescription("product1 Description");

product.setActive(**false**);

product.setSku("100 units");

product.setPrice(**new** BigDecimal(100));

product.setImageUrl("product.png");

Product p1 = repository.save(product);

System.***out***.println(p1);

}

@Test

**void** updateData() {

// find or retrieve an entity by Id

Long id = 1L;

Product product = repository.findById(id).get();

// Update entry information

product.setName("best product1");

product.setDescription("best product Description 1");

product.setImageUrl("raju.png");

// save entry in db

repository.save(product);

}

@Test

**void** findById() {

// find or retrieve an entity by Id

Long id = 1L;

Product product = repository.findById(id).get();

System.***out***.println(product.toString());

}

@Test

**void** saveAll() {

Product santoor = **new** Product();

santoor.setName("santoor");

santoor.setDescription("Indian brand");

santoor.setActive(**true**);

santoor.setSku("20 units");

santoor.setPrice(**new** BigDecimal(20));

santoor.setImageUrl("santoor.png");

Product lux = **new** Product();

lux.setName("Lux");

lux.setDescription("Us brand");

lux.setActive(**true**);

lux.setSku("60 units");

lux.setPrice(**new** BigDecimal(30));

lux.setImageUrl("Lux.png");

List<Product> arr = **new** ArrayList<>();

arr.add(lux);

arr.add(santoor);

repository.saveAll(arr);

}

@Test

**void** fetchAll() {

List<Product> allProduct = repository.findAll();

**for** (Product product : allProduct) {

System.***out***.println(product);

}

}

@Test

**void** deleteById() {

Long id = 52L;

repository.deleteById(id);

}

@Test()

**void** delete() {

Long id = 253L;

Product product = repository.findById(id).get();

repository.delete(product);

}

@Test

**void** deleteAll() {

repository.deleteAll();

}

@Test

**void** deleteAllById() {

Long id = 302L;

Long id2 = 303L;

Product product = repository.findById(id).get();

Product product2 = repository.findById(id2).get();

repository.deleteAll(List.*of*(product, product2));

}

@Test

**void** countMethod() {

Long count = repository.count();

System.***out***.println(count);

}

@Test

**void** existById() {

Long id = 359L;

**boolean** value = repository.existsById(id);

System.***out***.println(value);

}

@Test

**void** sortByMultipleFields() {

String sortBy = "name";

String sortByDesc = "description";

String sortDir = "desc";

Sort sortByName = sortDir.equalsIgnoreCase(Sort.Direction.***ASC***.name()) ? Sort.*by*(sortBy).ascending()

: Sort.*by*(sortBy).descending();

Sort sortByDescription = sortDir.equalsIgnoreCase(Sort.Direction.***ASC***.name()) ? Sort.*by*(sortByDesc).ascending()

: Sort.*by*(sortByDesc).descending();

Sort groupBySort = sortByName.and(sortByDescription);

List<Product> products = repository.findAll(groupBySort);

products.forEach((p) -> {

System.***out***.println(p);

});

}

step6: for Finding products bitween given Dates we need to add

**import** com.springdatajpa.springdatajpacourse.entity.Product;

**public** **interface** ProductRepository **extends** JpaRepository<Product, Long> {

List<Product>findByDateCreatedBetween(LocalDateTime startDate,LocalDateTime endDate);

}

@Test

**void** findByDate() {

LocalDateTime startDate = LocalDateTime.*of*(2023, 02, 19, 11, 32, 12);

LocalDateTime endDate = LocalDateTime.*of*(2023, 02, 19, 16, 29, 14);

List<Product> products = repository.findByDateCreatedBetween(startDate, endDate);

System.***out***.println("products length" + products.size());

products.forEach((p) -> {

System.***out***.println(p.getId());

System.***out***.println(p.getName());

});

}

<?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>3.0.2</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.springdatajpa</groupId>

<artifactId>spring-data-jpa-course</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-data-jpa-course</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>19</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- <https://mvnrepository.com/artifact/org.projectlombok/lombok> -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.10</version>

<scope>provided</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Some of the Custum Queries:

findByNameStartingWith("String prefix")

findByNameEndingWith("String suffix")

findByNameContaining("String words")

findByNameLike(String likePattern)

findByAgeLessThan(int age)

findByAgeGreaterThan(int age)

findByAgeIn(Collection<Integer>ages)

findByNameOrderBy(String Name)

Note:we need to write these method declerations inside the jpa repository interface and call them where ever required

if want to write our own Sql Query we can use @Query annotation

@Query("select u FROM User u)

public List<User> getAllUsers();

@Query("select u FROM User u WHERE u.name=:n and u.city=:c")

public List<User>getUserByName(@Param("n") String name,@Param("c")String city);

@Query(value ="select \* from user "nativeQuery=true)

public List<User>getUsers();

RestApi in Spring Boot

Get Request

Step1:First Create a Project and add Spring Web dependency

Step2: Then Create A Package descendent to main class package for Controller and Create a Java Class inside it

Step3: then Create a Model Class in entity Package

**package** com.rest.test.entity;

**public** **class** BookModel {

**private** **int** bookId;

**private** String bookName;

**private** String bookAuther;

**public** **int** getBookId() {

**return** bookId;

}

**public** **void** setBookId(**int** bookId) {

**this**.bookId = bookId;

}

**public** String getBookName() {

**return** bookName;

}

**public** **void** setBookName(String bookName) {

**this**.bookName = bookName;

}

**public** String getBookAuther() {

**return** bookAuther;

}

**public** BookModel(**int** bookId, String bookName, String bookAuther) {

**super**();

**this**.bookId = bookId;

**this**.bookName = bookName;

**this**.bookAuther = bookAuther;

}

**public** BookModel() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** **void** setBookAuther(String bookAuther) {

**this**.bookAuther = bookAuther;

}

}

Step4:Then Create a Fake Service Class For Fetching Data in real Case we need to Fetch From Database but now we Create a Dummy ArrayList

**package** com.rest.test.service;

**import** java.util.ArrayList;

**import** java.util.List;

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

**import** com.rest.test.entity.BookModel;

@Component

**public** **class** BookService {

**private** **static** List<BookModel> *bookArray* = **new** ArrayList<>();

**static** {

*bookArray*.add(**new** BookModel(12, "Vantala Book", "Tejas"));

*bookArray*.add(**new** BookModel(13, "Jokes Book", "Sharma"));

*bookArray*.add(**new** BookModel(14, "Fighting Book", "Rakesh"));

}

**public** List<BookModel> getAllBooks() {

**return** *bookArray*;

}

**public** BookModel getBookById(**int** id) {

BookModel bookModel = **null**;

bookModel = *bookArray*.stream().filter(e -> e.getBookId() == id).findFirst().get();

**return** bookModel;

}

}

we need add @Component because we want to use this class object later

Step5:Then we need add Controller class logic

@RestController

**public** **class** BookController {

@Autowired

BookService service;

@GetMapping("/books")

**public** List<BookModel> getBook() {

List<BookModel> book = service.getAllBooks();

**return** book;

}

@GetMapping("/book/{id}")

**public** BookModel getBookById(@PathVariable("id") **int** id) {

BookModel model = service.getBookById(id);

**return** model;

}

}

Post Request

step1: Add the Fallowing in RestControllers

@PostMapping("/book")

**public** BookModel createBook(@RequestBody BookModel book) {

BookModel b = service.createBook(book);

**return** b;

}

Step2:Add the Following Line in Service class

**public** BookModel createBook(BookModel b) {

*bookArray*.add(b);

**return** b;

}

Delete Request

Step1:Add the Fallowing in RestControllers

@DeleteMapping("/deletebook/{id}")

**public** **void** deleteBook(@PathVariable("id") **int** id) {

System.***out***.println("delete Called");

service.deleteBook(id);

}

Step2:Add the Following Line in Service class

**public** **void** deleteBook(**int** bookId) {

*bookArray*.stream().filter(e -> e.getBookId() != bookId).collect(Collectors.*toList*());

}

Update Request

Step1:Add the Fallowing in RestControllers

@PutMapping("/book/{id}")

**public** **void** updateBook(@PathVariable("id") **int** id, @RequestBody BookModel model) {

service.updateBook(id, model);

}

Step2:Add the Following Line in Service class

**public** **void** updateBook(**int** bookId, BookModel book) {

*bookArray* = *bookArray*.stream().map(b -> {

**if** (b.getBookId() == bookId) {

book.setBookAuther(book.getBookAuther());

book.setBookName(book.getBookName());

}

**return** book;

}).collect(Collectors.*toList*());

}

ResponseEntity class

@GetMapping("/books")

**public** ResponseEntity<List<BookModel>> getBook() {

List<BookModel> book = service.getAllBooks();

**if** (book.size() <= 0) {

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).build();

}

**return** ResponseEntity.*of*(Optional.*of*(book));

}

Rest Api With Spring Data Jpa

Step1:first we need to add spring data jpa and my sql conncector dependencies in pom.xml

step2:Then configure your database in application.properties

step3:Then Convert model class to Entity Model by Adding

@Entity and @Id Annotations

**package** com.rest.test.entity;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.GenerationType;

**import** jakarta.persistence.Id;

@Entity

**public** **class** BookModel {

@Id

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

**private** **int** bookId;

**private** String bookName;

**private** String bookAuther;

**public** **int** getBookId() {

**return** bookId;

}

**public** **void** setBookId(**int** bookId) {

**this**.bookId = bookId;

}

**public** String getBookName() {

**return** bookName;

}

**public** **void** setBookName(String bookName) {

**this**.bookName = bookName;

}

**public** String getBookAuther() {

**return** bookAuther;

}

**public** BookModel(**int** bookId, String bookName, String bookAuther) {

**super**();

**this**.bookId = bookId;

**this**.bookName = bookName;

**this**.bookAuther = bookAuther;

}

**public** BookModel() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** **void** setBookAuther(String bookAuther) {

**this**.bookAuther = bookAuther;

}

}

Step:4 Then Create your Repository Interface and extends Crud Repository

Step:5 Then Add Repository Reference in Service class and get the data using Db

**package** com.rest.test.service;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

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

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

**import** com.rest.test.dao.BookRepository;

**import** com.rest.test.entity.BookModel;

@Component

**public** **class** BookService {

**private** **static** List<BookModel> *bookArray* = **new** ArrayList<>();

@Autowired

BookRepository repository;

**public** List<BookModel> getAllBooks() {

List<BookModel> allBooks = (List<BookModel>) **this**.repository.findAll();

**return** allBooks;

}

**public** BookModel getBookById(**int** id) {

BookModel bookModel = **null**;

bookModel = *bookArray*.stream().filter(e -> e.getBookId() == id).findFirst().get();

**return** bookModel;

}

**public** BookModel createBook(BookModel b) {

repository.save(b);

**return** b;

}

**public** **void** deleteBook(**int** bookId) {

repository.deleteById(bookId);

}

**public** **void** updateBook(**int** bookId, BookModel book) {

BookModel book1 = repository.findById(bookId).get();

book1.setBookName(book.getBookName());

book1.setBookAuther(book.getBookAuther());

repository.save(book1);

}

}

One to One Custom Nested Model

Step1: create a nested Model means a class inside other class

in our example We are Creating Auther class inside

BookModel class

Step2:We need create Seperate Entity class For Auther

and Add @Entity and @Id Annotations Same as BookModel class

step3:Add below Annotation in BookModel class

@OneToOne(cascade = CascadeType.***ALL***)

**private** Auther bookAuther;

Step4:we are good to go now

Bidirectional Custom Nested Model

Step1:Add a BookModel variable in Auther entity

Step2:and then Add below two Annotations

@OneToOne(mappedBy = "bookAuther")

@JsonBackReference

**private** BookModel bookModel;

Step3:and Create getters and Setters for bookModel Variable

Step4:add one Additianal Annotation in BookModel class

@OneToOne(cascade = CascadeType.***ALL***)

@JsonManagedReference

**private** Auther bookAuther;

Step5:we are good to go

Serving Static Images in SpringBoot

Step1: enable multipart configuration in application.propeties File

spring.servlet.multipart.enabled=true

spring.servlet.multipart.max-file-size=300MB

spring.servlet.multipart.file-size-threshold=10KB

Step2:Create a RestContrller For FileUploading

**package** com.rest.test.controller;

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

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

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

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

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

**import** org.springframework.web.multipart.MultipartFile;

**import** org.springframework.web.servlet.support.ServletUriComponentsBuilder;

**import** com.rest.test.helper.FileUploadHelper;

@RestController

**public** **class** FileUploadController {

@Autowired

FileUploadHelper helper;

@PostMapping("/upload-file")

**public** ResponseEntity<String> uploadFile(@RequestParam("file") MultipartFile file) {

**if** (file.isEmpty()) {

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body("No File Uploaded");

}

helper.uploadFile(file);

**return** ResponseEntity.*ok*(ServletUriComponentsBuilder.*fromCurrentContextPath*().path("/images/")

.path(file.getOriginalFilename()).toUriString());

//this line is used to give the path till localhost:8080

//and Append remaining Folder till file path and Generate a Url

}

}

Note: (@RequestParam("file") the file is used as key while Uploading image from client Side value is our image

Step3:Then Creates Helper class package and a class inside it

for seperate the file uploading code from controller

**package** com.rest.test.helper;

**import** java.io.File;

**import** java.io.IOException;

**import** java.nio.file.Files;

**import** java.nio.file.Paths;

**import** java.nio.file.StandardCopyOption;

**import** org.springframework.core.io.ClassPathResource;

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

**import** org.springframework.web.multipart.MultipartFile;

@Component

**public** **class** FileUploadHelper {

**public** **final** String Upload\_Dir = **new** ClassPathResource("static/images/").getFile().getAbsolutePath();

**public** FileUploadHelper() **throws** IOException {

}

**public** **boolean** uploadFile(MultipartFile file) {

**boolean** f = **false**;

**try** {

Files.*copy*(file.getInputStream(), Paths.*get*(Upload\_Dir + File.***separator*** + file.getOriginalFilename()),

StandardCopyOption.***REPLACE\_EXISTING***);

**return** **true**;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** f;

}

}

Step4: after Uploading url is like this

<http://localhost:8080/images/car.jpg>

Jwt Autentication Spring

step1:Create a New Project and Add Required Dependencies in Pom.xml file

<?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>3.0.3</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.jwt</groupId>

<artifactId>auth</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>jwtAuth</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>19</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- <https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api> -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

<!-- <https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt> -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.8.0</version>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Step2:Create a Controller for Welcome Screen

package com.jwt.auth.controller;

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

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

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

@RestController

public class Home {

@RequestMapping(value = "/welcome",method=RequestMethod.GET)

public String welcome() {

System.out.println("Welcome page called");

return "This is welcome Page";

}

}

Step3: Create a configuration class to Confure Which urls we need allow publicly and which we need to ristrict and enable disable cors and adding Filters etc

package com.jwt.auth.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.password.NoOpPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import com.jwt.auth.services.CustomUserDetailsService;

@Configuration

@EnableWebSecurity

public class MyConfig {

@Autowired

CustomUserDetailsService customUserDetailsService;

@Autowired

private JwtAuthenticationFilter jwtAuthenticationFilter;

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeHttpRequests().requestMatchers("/token").permitAll()

.anyRequest().authenticated().and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

@SuppressWarnings("deprecation")

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(customUserDetailsService);

authProvider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());

return authProvider;

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {

return authConfig.getAuthenticationManager();

}

@SuppressWarnings("deprecation")

@Bean

public PasswordEncoder passwordEncoder() {

return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();

}

}

Step4: We need to Create CustomUserDetails Service class to

get the UserDetails with username

package com.jwt.auth.services;

import java.util.ArrayList;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

@Service

public class CustomUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

if(username.equals("durgesh")) {

return new User("durgesh","durgesh",new ArrayList<>());

}

else {

throw new UsernameNotFoundException("user Not Found");

}

}

}

Step5:We need to Create a Jwt Util class is used to which will help us all token related operations

package com.jwt.auth.helper;

import java.util.Date;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Service

@Component

public class JwtUtil {

private String SECRET\_KEY = "secret";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

private Boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return createToken(claims, userDetails.getUsername());

}

private String createToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY).compact();

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

Step6:Then we need to create two Models one for while requesting and one for to send Resonce back to User

package com.jwt.auth.models;

public class JwtRequest {

private String username;

private String password;

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

@Override

public String toString() {

return "JwtRequest [username=" + username + ", password=" + password + "]";

}

public JwtRequest() {

}

public JwtRequest(String username, String password) {

this.username = username;

this.password = password;

}

}

package com.jwt.auth.models;

public class JwtResponse {

private String token;

public JwtResponse(String token) {

this.token = token;

}

public void setToken(String token) {

this.token = token;

}

@Override

public String toString() {

return "JwtResponse [token=" + token + "]";

}

}

Step 7:The most importent step is to Create a Controller generating Jwt token

package com.jwt.auth.controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

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

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

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

import com.jwt.auth.helper.JwtUtil;

import com.jwt.auth.models.JwtRequest;

import com.jwt.auth.models.JwtResponse;

import com.jwt.auth.services.CustomUserDetailsService;

@RestController

class JwtController {

@Autowired

CustomUserDetailsService userDetailsService;

@Autowired

JwtUtil jwtUtil;

@Autowired

DaoAuthenticationProvider auth;

@PostMapping(value = "/token")

public ResponseEntity<?> generateToken(@RequestBody JwtRequest jwtRequest) throws Exception {

try {

System.out.println(jwtRequest);

auth.authenticate(

new UsernamePasswordAuthenticationToken(jwtRequest.getUsername(), jwtRequest.getPassword()));

System.out.println("auth succes");

System.out.println(jwtRequest);

} catch (UsernameNotFoundException exception) {

exception.printStackTrace();

throw new Exception("Bad Credentials");

}

UserDetails userDetails = userDetailsService.loadUserByUsername(jwtRequest.getUsername());

String token = jwtUtil.generateToken(userDetails);

System.out.println("token generated"+token);

return ResponseEntity.ok(token);

}

}

In this First we need to get the username and Password from the Request

and then we need to Authenicate them

auth.authenticate(

new UsernamePasswordAuthenticationToken(jwtRequest.getUsername(), jwtRequest.getPassword()));

with this line

and then load the userDetails From the database with the Help of username

and generate token with those details

UserDetails userDetails = userDetailsService.loadUserByUsername(jwtRequest.getUsername());

String token = jwtUtil.generateToken(userDetails);

Step: 8 Ofter Creating Token then we need create JwtFilter class To verify the token is valid or not For every Request

package com.jwt.auth.config;

import java.io.IOException;

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import com.jwt.auth.helper.JwtUtil;

import com.jwt.auth.services.CustomUserDetailsService;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@Component

class JwtAuthenticationFilter extends OncePerRequestFilter {

@Autowired

JwtUtil jwtUtil;

@Autowired

CustomUserDetailsService userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

// get Jwt

// Bearer

// validate

String requestTokenHeader = request.getHeader("Authorization");

String userName = null;

String jwtToken = null;

// null and Format

if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {

jwtToken = requestTokenHeader.substring(7);

try {

userName = jwtUtil.extractUsername(jwtToken);

} catch (Exception e) {

e.printStackTrace();

}

UserDetails userDetails = userDetailsService.loadUserByUsername(userName);

// security

if (userName != null && SecurityContextHolder.getContext() != null) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

} else {

System.out.println("Token Not Validate");

}

}

filterChain.doFilter(request, response);

}

}

Step:9 We need to Add this line ofter Creating Filter inside Configuration file

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

and below method

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeHttpRequests().requestMatchers("/token").permitAll()

.anyRequest().authenticated().and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

Printing Present Working Directory

String cwd = Path.of("").toAbsolutePath().toString();

System.out.println("path value is"+cwd);

Moving Files From TempDirectory To resource Folder

@GetMapping("/fileTest/{path}")

public void testFile(@PathVariable String path) throws IOException {

File tmpDir = new File("C:\\Users\\DELL\\Pictures\\Camera Roll");

File checkList = new File(tmpDir + File.separator + path);

System.out.println(checkList);

if (checkList.exists()) {

File sourceFile = checkList;

File target = new File(new ClassPathResource("static/images/").getFile().getAbsolutePath());

File targetFile = new File(target + File.separator + path);

System.out.println("targetFile" + targetFile);

Files.move(sourceFile.toPath(), targetFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);

Basic Authenitication In SpringBoot

Step1:add the spring security dependency

<dependency>

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

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Step2:add the username,password and role in application.properties file

spring.security.user.name=rakesh

spring.security.user.password=rakesh12

spring.security.user.roles=ADMIN

Step3: Then create Config file and add basic auth in that file

@EnableWebSecurity

@Configuration

public class SecurityConfig {

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeHttpRequests().and().httpBasic();

return http.build();

}

Basic Authenitication In SpringBoot

With DataBase

Step1:add the spring security dependency

<dependency>

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

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Step2:create entity for role

package com.codewithdurgesh.blog.entities;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

private 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;

}

}

Step3:Map the User With Role

@ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)

@JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user", referencedColumnName = "id"), inverseJoinColumns = @JoinColumn(name = "role", referencedColumnName = "id"))

private Set<Role> roles = new HashSet<>();

@Entity

@Table(name = "users")

public class User implements UserDetails {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;

    @Column(name = "user\_name", length = 200, nullable = false)

    private String name;

    private String email;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

    private List<Post> posts = new ArrayList<>();

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)

    @JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user", referencedColumnName = "id"), inverseJoinColumns = @JoinColumn(name = "role", referencedColumnName = "id")

    )

    private Set<Role> roles = new HashSet<>();

    public void setId(int id) {

        this.id = id;

    }

    public User() {

    }

    public void setName(String name) {

        this.name = name;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    public void setAbout(String about) {

        this.about = about;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    private String about;

    private String password;

    public int getId() {

        return id;

    }

    public String getName() {

        return name;

    }

    public String getEmail() {

        return email;

    }

    public String getAbout() {

        return about;

    }

    public String getPassword() {

        return password;

    }

    @Override

    public Collection<? extends GrantedAuthority> getAuthorities() {

        List<SimpleGrantedAuthority> authorities = this.roles.stream()

                .map((role) -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());

        return authorities;

    }

    @Override

    public String getUsername() {

        return this.email;

    }

    @Override

    public boolean isAccountNonExpired() {

        return true;

    }

    @Override

    public boolean isAccountNonLocked() {

        return true;

    }

    @Override

    public boolean isCredentialsNonExpired() {

        return true;

    }

    @Override

    public boolean isEnabled() {

        return true;

    }

}

}

Step4:Then Create a Custum userDetails Service class

package com.codewithdurgesh.blog.security;

import java.util.Optional;

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

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import com.codewithdurgesh.blog.entities.User;

import com.codewithdurgesh.blog.exceptions.ResourseNotFoundException;

import com.codewithdurgesh.blog.repositories.UserRepository;

@Service

public class CustomUserDetailsService implements UserDetailsService {

@Autowired

private UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByEmail(username)

.orElseThrow(() -> new ResourseNotFoundException("User", "Id", 0));

return user;

}

}

Step5:Add this line in user repository

public interface UserRepository extends JpaRepository<User, Integer> {

Optional<User> findByEmail(String email);

}

Step6:Encode the password in main

@SpringBootApplication

public class BlogApplication implements CommandLineRunner {

@Autowired

private PasswordEncoder passwordEncoder;

public static void main(String[] args) {

SpringApplication.run(BlogApplication.class, args);

}

@Bean

public ModelMapper mapper() {

return new ModelMapper();

}

@Override

public void run(String... args) throws Exception {

String code = passwordEncoder.encode("abc");

System.out.println(code);

}

}

Step7:Add the password encoder in config file

package com.codewithdurgesh.blog.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import com.codewithdurgesh.blog.security.CustomUserDetailsService;

@EnableWebSecurity

@Configuration

public class SecurityConfig {

@Autowired

private CustomUserDetailsService userDetailsService;

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeHttpRequests().and().httpBasic();

return http.build();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {

return authConfig.getAuthenticationManager();

}

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService);

authProvider.setPasswordEncoder(passwordEncoder());

return authProvider;

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

D:\projects\testprojects\javaprojects\bootrestexample\src\main\resources\static\images

D:\projects\testprojects\javaprojects\bootrestexample\target\classes\static\images

File sourcePath = new File(

"D:\\projects\\testprojects\\javaprojects\\bootrestexample\\target\\classes\\static\\temp\\car.jpg");

File targetPath = new File(

"D:\\projects\\testprojects\\javaprojects\\bootrestexample\\target\\classes\\static\\images\\car.jpg");

flutter\_offline: "^2.1.0"

File sourceFile = new File("C:\\Users\\DELL\\Pictures\\Screenshots\\rebase.png");

File targetFile = new File("C:\\Users\\DELL\\Pictures\\Camera Roll\\rebase.png");

Files.move(sourceFile.toPath(), targetFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);

package com.rest.test.helper;

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardCopyOption;

import java.util.Random;

import java.util.random.RandomGenerator;

import javax.print.event.PrintEvent;

import org.springframework.core.io.ClassPathResource;

import org.springframework.stereotype.Component;

import org.springframework.web.multipart.MultipartFile;

@Component

public class FileUploadHelper {

public final String Upload\_Dir = new ClassPathResource("static/temp/").getFile().getAbsolutePath();

// public final String Upload\_Dir=

// "D:\\projects\\testprojects\\javaprojects\\bootrestexample\\src\\main\\resources\\static\\images";

public FileUploadHelper() throws IOException {

}

public boolean uploadFile(MultipartFile file) {

boolean f = false;

try {

File tmpDir = new File(new ClassPathResource("static/temp/").getFile().getAbsolutePath());

File checkList = new File(tmpDir + "/car.jpg");

File sourcePath = new File(new ClassPathResource("static/temp/car.jpg").getFile().getAbsolutePath());

File targetPath = new File(new ClassPathResource("static/images/car.jpg").getFile().getAbsolutePath());

System.out.println(targetPath);

boolean exists = checkList.exists();

System.out.println(exists);

String randomString = generateString();

System.out.println("random String value" + randomString);

if (exists) {

Files.move(Paths.get(sourcePath.getAbsolutePath()), Paths.get(targetPath.getAbsolutePath()),

StandardCopyOption.REPLACE\_EXISTING);

}

Files.copy(file.getInputStream(),

Paths.get(Upload\_Dir + File.separator + file.getOriginalFilename()),

StandardCopyOption.REPLACE\_EXISTING);

return true;

} catch (Exception e) {

e.printStackTrace();

}

return f;

}

public static String generateString() {

Random ran = new Random();

int top = 2;

char data = 'h';

String dat = "x";

for (int i = 0; i <= top; i++) {

data = (char) (ran.nextInt(25) + 97);

dat = data + dat;

}

return dat;

}

}

package com.rest.test.controller;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

import org.springframework.web.multipart.MultipartFile;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.rest.test.helper.FileUploadHelper;

@RestController

public class FileUploadController {

@Autowired

FileUploadHelper helper;

@PostMapping("/upload-file")

public ResponseEntity<String> uploadFile(@RequestParam("file") MultipartFile file) {

if (file.isEmpty()) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("No File Uploaded");

}

helper.uploadFile(file);

return ResponseEntity.ok(ServletUriComponentsBuilder.fromCurrentContextPath().path("/images/")

.path(file.getOriginalFilename()).toUriString());

// this line is used to give the path till localhost:8080

// and Append remaining Folder till file path and Generate a Url

}

}

OOPS CONCEPT

Datahiding:

Outside persion can’t access our internal data directly or our internal data should not go out directly this oop feature is nothing but data hiding.ofter validation only out side persion can access our internal data.

* By declaring data member as private we can achieve data hiding.

Ex:

public class Account{

private double balance;

public double getBalance(){

return balance;

}

}