
UNIT 8 SPRING MVC WITH BOOTSTRAP CSS

- 8.0 Introduction
- 8.1 Objectives
- 8.2 Configuration of Bootstrap in Spring Application
 - 8.2.1 Different ways to configure Bootstrap
- 8.3 Apply custom CSS in pages
- 8.4 Setting UP Database using Hibernate
 - 8.4.1 Java based configuration for Spring and Hibernate Integration
- 8.5 Create, Read, Update, and Delete (CRUD)
 - 8.5.1 Mapping CRUD to SQL Statements
 - 8.5.2 Mapping CRUD to REST
- 8.6 CRUD examples in Spring MVC and Hibernate
- 8.7 Summary
- 8.8 Solutions/ Answer to Check Your Progress
- 8.9 References/Further Reading

8.0 INTRODUCTION

Nowadays, huge traffic comes from mobile devices, and most of the internet searches like google are done on mobile devices, whether smartphones or tablets. The internet users who search a company to buy food or items, visit the company's website from the google link on the mobile devices themselves. If a company's website is not flexible across all screen resolutions and devices, it risks missing out on a large group of buyers. A responsive website prevents this loss and leads to substantial revenue gains.

Responsive web design improves user experience and creates a positive perception for a company or business. If your customers can access your website easily on all platforms, they're more likely to return to you for more business in the future.

It is a big challenge for website developers to keep the design responsive as the web evolves more and more. Bootstrap is the solution to this big challenge and makes things a whole lot easier. Bootstrap takes care of everything related to responsiveness without the user need to do the responsive bit. Bootstrap automatically scales in and out among the devices such as mobile phones, tablets, laptops, desktop computers, screen readers, etc.

Spring is the most popular and widely used Java EE framework, while Hibernate is the most popular ORM framework to interact with databases. Spring supports integrating Hibernate very easily, which makes the Hibernate the first choice as an ORM with Spring.

CRUD is an acronym for CREATE, READ, UPDATE and DELETE operations which are used in relational database applications and executed by mapping to a SQL statement or by standard HTTP verbs such as POST, GET, PUT, DELETE. CRUD is thus data-oriented and uses standardized HTTP action verbs.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Describe responsive web design and show its advantages,

- use Bootstrap integration with Spring MVC,
- apply different approaches to include Bootstrap into Spring MVC,
- include custom CSS into Spring applications,
- do Spring and Hibernate integration,
- do CRUD execution mapping to SQL and Rest Web Service, and
- develop a complete Spring MVC web application with Bootstrap, custom CSS and Hibernate.

8.2 CONFIGURATION OF BOOTSTRAP IN SPRING APPLICATION

Bootstrap was originally created by a designer and developer at Twitter in mid 2010. Prior to being an open-sourced framework, Bootstrap was known as *Twitter Blueprint*. **Bootstrap** is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development. Unlike many web frameworks, it concerns itself with front-end development only. It contains HTML and CSS based design templates for various interface components such as typography, forms, buttons, modals, image carousels, navigation etc., as well as optional JavaScript extensions. Bootstrap provides the ability to create responsive designs faster and easily. **Responsive Web Design (RWD)** is an approach to web design that makes web pages adjust automatically to look good and render well on a variety of devices and window or screen sizes from minimum to maximum display size. The various factors which support learning Bootstrap are as follows.

- **Browser Support:** Bootstrap supports all popular web browsers such as Chrome, Safari, Firefox, Opera, and Internet edge.
- **Responsive Design:** Bootstrap enables us to write responsive websites very easily. Its responsive CSS adjusts automatically to Desktops, Tablets and Mobiles.



- **Uniform solution to build interface:** Bootstrap provides a clean and uniform solution to build an interface for the developers.
- **Customization:** Bootstrap provides a simple and easy way to customize the CSS.
- **Functional built-in components:** Bootstrap contains beautiful and functional built-in components which can be customized very easily.

8.2.1 Different ways to configure Bootstrap

The Bootstrap 5 is the most recent version of the Bootstrap framework. There are two ways to include Bootstrap into JSP or HTML. The required configurations for Bootstrap 4 and Bootstrap 5 has been described as follows.

1. Bootstrap through Content Delivery Network (CDN)

Bootstrap CDN is an efficient and faster way to deliver the content from your website to the users. Bootstrap CDN speeds up the websites as CDN servers are intelligent enough to serve the resources based on proximity.

Including Bootstrap 4 into JSP/HTML.

For some of the functionalities Bootstrap 4 JavaScript file requires jQuery and Popper JavaScript, and these two must be loaded before loading of bootstrap.min.js file.

- **CSS:** Include the below link to the <head> tag of your desired HTML/JSP file.

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/boots
trap.min.css" integrity="sha384-
gg0YR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
```

- **JS:** Place the following <script>s near the end of your pages, right before the closing </body> tag, to enable them. jQuery must come first, then Popper.js, and then our JavaScript plugins.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
```

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/p
opper.min.js" integrity="sha384-
U02eT0CpHqdsJQ6hJty5KVphtPhzWj9W01c1HTMga3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
```

```
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstr
ap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEeAff/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

Including Bootstrap 5 into JSP/HTML

Bootstrap 5 no longer needs jQuery as a dependency since JavaScript can provide the same functionality. Add the following CSS and JS through Bootstrap 5 CDN to JSP/HTML:

- **CSS:** Copy-paste the below Stylesheet link to the <head> tag of your desired HTML/JSP file.

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootst
rap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWFSpd3yD65VohhpuuCOMLASjC"
crossorigin="anonymous">
```

- **JS:** Place the following <script>s near the end of your pages, right before the closing </body> tag.

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/po
pper.min.js" integrity="sha384-
IQsolX15PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstra
p.min.js" integrity="sha384-
cVKIPhGWic2Al4u+LWgxFKTRicfu0JTtxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
```

2. Custom configuration of Bootstrap with Spring MVC (Offline by downloading files locally)

Another approach to include Bootstrap is to directly download the Bootstrap CSS and JS files locally to the project folder and then include the local copy into JSP/HTML. The following links can be used to download the Bootstrap.

- Bootstrap 4: <https://getbootstrap.com/docs/4.3/getting-started/download/>
- Bootstrap 5: <https://v5.getbootstrap.com/docs/5.0/getting-started/download/>

Download the Bootstrap and perform the following steps to configure Bootstrap in the Spring application.

- Put static resources CSS and JS files into webapp/resources/static directory. To segregate CSS and JS, keep them into respective folders webapp/resources/static/js and webapp/resources/static/css into the eclipse project.
- Create the resource mapping into Spring using either XML configuration or Java configuration.

XML configuration

```
<mvc:resources mapping="/js/**" location="/resources/static/js/" />
<mvc:resources mapping="/css/**" location="/resources/static/css/" />
```

Java configuration

```
@Configuration
@EnableWebMvc
public class MyApplication implements WebMvcConfigurer
{
    public void addResourceHandlers(final ResourceHandlerRegistry registry)
    {
        registry.addResourceHandler("/js/**").addResourceLocations("/resources/static/js/");
        registry.addResourceHandler("/css/**").addResourceLocations("/resources/static/css/");
    }
}
```

- Include the Bootstrap CSS and JS into JSP page via JSTL tag **c:url** or Spring tag **spring:url**.
JSTL tag **c:url**

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="en">
<head>
<link href="<c:url value="/css/bootstrap.css" />" rel="stylesheet">
<script src="<c:url value="/js/jquery.1.10.2.min.js" />"></script>
<script src="<c:url value="/js/bootstrap.js" />"></script>
</head>
<body>
</body>
</html>
```

Spring tag **spring:url**

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<spring:url value="/css/bootstrap.css" var="bootstrapCss" />
<spring:url value="/js/jquery.1.10.2.min.js" var="jqueryJs" />
<spring:url value="/js/bootstrap.js" var="bootstrapJs" />

<link href="${bootstrapCss}" rel="stylesheet" />
<script src="${jqueryJs}"></script>
<script src="${bootstrapJs}"></script>
</head>
<body><body>
</body>
</html>
```

Check Your Progress 1

- 1) What is Bootstrap and what factors make Bootstrap a popular choice to use in web applications?

.....

.....

.....

.....

- 2) What are the different ways to configure Bootstrap in Spring MVC?

.....

.....

.....

.....

- 3) What are the obvious reasons to use Bootstrap CDN to configure it with Spring?

.....

.....

.....

.....

8.3 APPLY CUSTOM CSS IN PAGES

CSS or Cascading Style Sheets is a method of adding stylistic instructions for your website to its back-end code. The HTML (hypertext markup language) is the most common coding language for a website. CSS is an extension of HTML that outlines specific stylistic instructions to style websites. It allows you to customize the look of your website and apply stylistic decisions across its entirety. It also allows you to separate the style from the structure of a web page. CSS is used to specify the color of

a heading or what font your content should be written in. CSS can be embedded into HTML elements, as shown below.

```
<p style="color: green;">
  This is Green Color Text.
</p>
```

The above changes the style of the text for a particular <p> tag. This style can be separated from <p> tag with following instructions.

```
<head>
<style type="text/css">
  p
  {
    color: green;
  }
</style>
</head>
<body>
<p>
  This is Green Color Text.
</p>
<p>
  This is another paragraph with a green color.
</p>
</body>
```

The above separated embedded style instructions change all the text within <p> tags **on a page** to green. A website generally consists of multiple pages, and the pages should be uniform and consistent in terms of font-size, font-color, font-family etc. To keep the style uniform across the pages, a separate CSS file with any name e.g. custom.css can be created, and custom styles can be added to this file. The required configuration for custom css, in Spring MVC is like using the Bootstrap downloading the files locally described into section 8.2.1. The custom css can be included at the top of each webpage inside the <head> tag using the below instruction.

```
<link rel="stylesheet" type="text/css" href="css/custom.css">
```

There are three main components that must be understood clearly to proceed with adding customized CSS language to the website. The components are **selectors, properties and values**.

The selector uses HTML to identify the part of your website that you want to style. For example, the HTML code for a paragraph is “p” and if you want to use CSS to change the style of the paragraph, “p” becomes your selector.

Properties and values are then used to apply stylistic instructions to the selectors. If you want your paragraphs to be written in red text, the property will come first and will indicate the specific attribute that you’re trying to change (**color in this case**). The value is what you want to change that attribute to, which is **red** in our example.

The section 8.6 describes the CRUD application using Spring MVC and Hibernate. Custom CSS and configuration code have been described there. Check the impact of custom CSS on the following screens.

Student Management System (SMS)				
Registered Students				
Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete
Add New Student				

Figure 8.1: CRUD Application Page Using Bootstrap

Student Management System (SMS)				
Registered Students				
Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete
Add New Student				

Figure 8.2: CRUD Application Page Using Bootstrap And Custom CSS

8.4 SETTING UP DATABASE USING HIBERNATE(Database configuration using Hibernate)

Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate. The database setup can be done with a few lines of Java code or XML configuration. Spring provides two key beans to support the Hibernate integration, available in the org.springframework.orm.hibernate5 package:

- **LocalSessionFactoryBean:** creates a Hibernate's **SessionFactory** which is injected into Hibernate-based DAO classes.
- **PlatformTransactionManager:** provides transaction support code for a **SessionFactory**. Programmers can use **@Transactional** annotation in DAO methods to avoid writing boiler-plate transaction code explicitly.

The following section describes how these concepts are implemented in a real project with the required maven dependencies and Java configuration for Hibernate.

Maven Dependencies

The necessary dependencies for pom.xml to integrate Hibernate with Spring are as follows.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.0.RELEASE</version>
</dependency>

<!--Hibernate -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.5.Final</version>
</dependency>

<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.16</version>
</dependency>
```

8.4.1 Java based configuration for Spring and Hibernate Integration

The required beans such as LocalSessionFactoryBean, DataSource, and PlatformTransactionManager, as well as some Hibernate-specific properties have been defined into HibernateConfig.

```
package com.ignou.mvcapp;
import java.util.Properties;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;
```

```
@Configuration
@EnableTransactionManagement
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfig
{
    @Autowired
    private Environment environment;

    @Bean
    public LocalSessionFactoryBean sessionFactory()
    {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan(new String[] { "com.ignou.mvcapp.model" });
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }

    @Bean
    public DataSource dataSource()
    {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
```



```

dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));
dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));
dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));
dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));
return dataSource;
}

private Properties hibernateProperties()
{
    Properties properties = new Properties();
    properties.put("hibernate.dialect", environment.getProperty("hibernate.dialect"));
    properties.put("hibernate.show_sql", environment.getProperty("hibernate.show_sql"));
    properties.put("hibernate.format_sql", environment.getProperty("hibernate.format_sql"));
    properties.put("hibernate.hbm2ddl.auto", environment.getProperty("hibernate.hbm2ddl.auto"));
    return properties;
}

@Bean
public PlatformTransactionManager getTransactionManager()
{
    HibernateTransactionManager htm = new HibernateTransactionManager();
    htm.setSessionFactory(sessionFactory().getObject());
    return htm;
}
}

```

Property file inside src/main/resources. File name can be any with extension as properties.

```

jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/schoolDB
jdbc.username = root
jdbc.password = root

hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true
hibernate.hbm2ddl.auto = update

```

☛ Check Your Progress 2

- 1) What is custom CSS and how can these be included into a JSP?
.....
.....
.....
- 2) Write the custom CSS to make all the paragraphs of a page to red color. Write the sample JSP page for it.
.....
.....
.....
.....
- 3) What are the key beans provided by Spring to integrate with Hibernate?
.....
.....
.....
.....

- 4) Write the skeleton code for Hibernate Configuration code into Spring.

.....
.....
.....
.....

8.5 CREATE, READ, UPDATE, AND DELETE (CRUD)

Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE. The CRUD paradigm is common in constructing web applications. While constructing the APIs, the model should provide four basic types of functionality such as Create, Read, Update and Delete the resources. CRUD operations are also often used with SQL.

It can also describe user-interface conventions that allow viewing, searching and modifying information through computer-based forms and reports. Most applications have some form of CRUD functionality. Let's consider that we have a student database that stores information for students and grades. When programmers provide interactions with this database (often through stored procedures), the steps of CRUD are carried out as follows:

- Create: A new student is entered into the database.
- Read: The student's information is displayed to the users.
- Update: Already, existing student's attributes are being updated with new values.
- Delete: If the student is not part of the institute, the student can be deleted from the records.

8.5.1 Mapping CRUD to SQL Statements

The previous section explained the concept of **CRUD** and defined it based on different contexts. The following table maps **CRUD to database sql** operation with an example.

Function	SQL Statement	Example
C(reate)	Insert	Insert into student (name,grade) values('Prasoon',10);
R(ead)	Select	Select * from student
U(pdate)	Update	Update student set grade=10 where id=1;
D(elete)	Delete	Delete from student where id=1;

8.5.2 Mapping CRUD to REST

In **REST context**, CRUD corresponds to Rest web service **POST, GET, PUT and DELETE** respectively. The following table maps CRUD to REST with example.

Function	Rest web service	Example
C(reate)	POST	POST http://teststudent.com/students/
R(ead)	GET	GET http://teststudent.com/students/
U(pdate)	PUT	PUT http://teststudent.com/students/1
D(elete)	DELETE	DELETE http://teststudent.com/students/1

The following sections explain the CRUD operations in the REST environment for the system, Student Management System (SMS), to keep the student records with their grade.

... Create

To create a resource in a REST environment, HTTP Post method is used. Post method creates a new resource of the specified resource type. To create a new student record for the Student Management System, HTTP POST request and response are shown below.

- **Request:** POST <http://teststudent.com/students/>

- **Request Body:**

```
{  
  "firstName": "Anurag",  
  "lastName": "Singh",  
  "grade": 10  
}
```

- **Response:** Status Code – 201(Created)

```
{  
  "id": 1,  
  "firstName": "Anurag",  
  "lastName": "Singh",  
  "grade": 10  
}
```

... Read

To read resources in a REST environment, HTTP GET method is used. Read operation should not change the resource. The Get method returns the same response irrespective of number of times of execution. HTTP GET method is **idempotent**. A HTTP method is called idempotent if an identical request can be made once or several times in a row with the same effect while leaving the server in the same state. To retrieve all student records from the Student Management System, HTTP GET method request and response is shown below.

- **Request:** GET <http://teststudent.com/students/>

- **Response:** Status Code – 200 (OK)

```
[  
  {"id": 1, "firstName": "Anurag", "lastName": "Singh", "grade": 10},  
  {"id": 1, "firstName": "Dinesh", "lastName": "Chaurasia", "grade": 10}  
]
```

... Update

To update a resource in a REST environment, HTTP **PUT or PATCH** method is used. This operation is idempotent. Request and response to update the grade of student id 1, is shown below.

- **Request:** PUT <http://teststudent.com/students/1>

- **Request Body:**

```
{  
  "id": 1,  
  "firstName": "Anurag",  
  "lastName": "Singh",  
  "grade": 8  
}
```

- **Response:** Status Code – **200 (OK)**. The response includes a Status Code of **200 (OK)** to signify that the operation was successful, but it need not return a response body.

... Delete

To delete a resource in a REST environment, **HTTP DELETE** method is used. It is used to remove a resource from the system. This operation is also **idempotent**. Request and response to delete a student with id 1, is shown below.

- Request: DELETE http://teststudent.com/students/1
- **Response:** Status Code – **204 (No Content)**. Successful delete operation returns a response code of **204 (NO CONTENT)**, with no response body.

8.6 CRUD EXAMPLES IN SPRING MVC AND HIBERNATE

The previous section has explained the CRUD operation for the REST environment. This section is full of code to integrate **Spring MVC** and **Hibernate** to implement the **CRUD** operation at the database level using **MySQL** database. Given below is described the Student Management System application implements which were explained in previous sections. These include the following:

- Bootstrap configuration into Spring MVC using CDN
- Custom CSS
- Spring and Hibernate integration
- CRUD operation implementation with MySQL

The required tools and software to complete the implementations of Student Management System application are as follows:

- Eclipse IDE
- Maven
- Tomcat 9
- MySQL 8
- Java 8 or Higher

Perform the following steps to get a good hands-on in **Spring MVC and Hibernate integration along with Bootstrap and custom CSS**.

Step 1: Create a maven project for a web application using the below maven command and import the project into eclipse. Maven project creation is described in unit 6 section 6.7.

```
mvn archetype:generate -DgroupId=com.ignou.springcrud -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4 -DartifactId=springmvc-crud-app -  
DinteractiveMode=false
```

If you get any error like missing folder into eclipse project, go to Java Build Path into project properties. Select the JRE and Maven Dependencies into Order and export tab. Project folder structure is shown below.

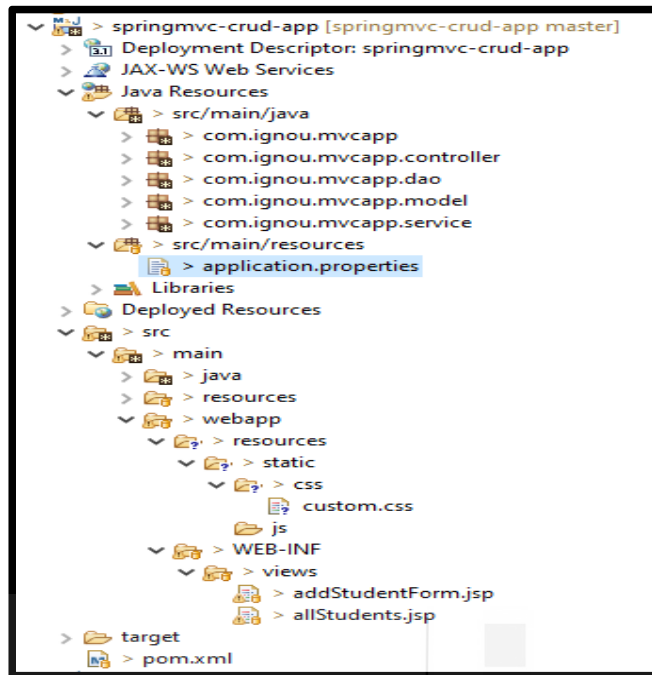


Figure 8.3: CRUD Application Folder Structure In Eclipse

Add the maven dependencies into pom.xml, which is given in section 8.4

Step2: Create the configuration classes into the package **com.ignou.mvcapp**
WebMvcConfig.java

```
package com.ignou.mvcapp;
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.ignou.mvcapp")
public class WebMvcConfig implements WebMvcConfigurer {
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("messages");
        return messageSource;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/js/**").addResourceLocations("/resources/static/js/");
        registry.addResourceHandler("/css/**").addResourceLocations("/resources/static/css/");
        registry.addResourceHandler("/images/**").addResourceLocations("/resources/static/images/");
    }
}
```

HibernateConfig.java

The Hibernate integration with Spring MVC configuration code has been given into section 8.4.1.

AppInitializer.java

```
package com.ignou.mvcapp;
public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer
{

    @Override
    protected Class<?>[] getRootConfigClasses()
    {
        return new Class[] { HibernateConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses()
    {
        return new Class[] { WebMvcConfig.class };
    }

    @Override
    protected String[] getServletMappings()
    {
        return new String[] { "/" };
    }
}
```

Step 3: Create the controller into package **com.ignou.mvcapp.controller**

AppController.java

```
package com.ignou.mvcapp.controller;
@Controller
@RequestMapping("/")
public class AppController
{

    @Autowired
    StudentService studentService;

    @Autowired
    MessageSource messageSource;

    /**
     * This Method will list All existing Students
     */
    @RequestMapping(value = { "/", "/list" }, method = RequestMethod.GET)
    public ModelAndView welcome(ModelMap model)
    {
        ModelAndView mv = new ModelAndView();
        List<Student> list = studentService.listAllStudents();
        model.addAttribute("allstudents", list);
        mv.setViewName("allStudents");
        return mv;
    }

    /**
     * This Method will provide way to Add New Student
     */
    @RequestMapping(value = { "/new" }, method = RequestMethod.GET)
    public String newStudentForm(ModelMap model)
    {
        Student newStudent = new Student();
        model.addAttribute("student", newStudent);
        model.addAttribute("edit", false);
        return "addStudentForm";
    }

    @RequestMapping(value = { "/new" }, method = RequestMethod.POST)
    public ModelAndView saveStudent(@Valid Student student, BindingResult result, ModelMap model)
    {
        ModelAndView mv = new ModelAndView();
        Integer grade = student.getGrade();
        if (grade == null)
        {
            model.addAttribute("student", student);
            model.addAttribute("error", "Grade must be Numeric.");
            mv.setViewName("addStudentForm");
        }
    }
}
```

```

    }
    else
    {
        studentService.saveStudent(student);
        List<Student>list = studentService.listAllStudents();
        model.addAttribute("allstudents", list);
        model.addAttribute("success", "Student " + student.getName() + " added successfully.");
        mv.setViewName("allStudents");
    }
    returnmv;
}

/**
 * This Method will provide the way to update an existing Student
 *
 * @param id
 * @param model
 * @return
 */
@RequestMapping(value = { "/edit-{id}" }, method = RequestMethod.GET)
public String editStudent(@PathVariable Long id, ModelMapmodel)
{
    Student student = studentService.getStudent(id);
    model.addAttribute("student", student);
    model.addAttribute("edit", true);
    return"addStudentForm";
}

/**
 * This method will be called on form submission, handling POST request for
 * updating Student in database. It also validates the user input
 *
 * @param Student
 * @param result
 * @param model
 * @param id
 * @return
 */
@RequestMapping(value = { "/edit-{id}" }, method = RequestMethod.POST)
public ModelAndViewupdateStudent(@Valid Student student, BindingResultresult, ModelMapmodel,
    @PathVariable Long id)
{
    ModelAndViewmv = newModelAndView();
    Integer grade = student.getGrade();
    if (grade == null)
    {
        model.addAttribute("student", student);
        model.addAttribute("edit", true);
        model.addAttribute("error", "Grade must be Numeric.");
        mv.setViewName("addStudentForm");
    }
    else
    {
        studentService.update(id, student);
        List<Student>list = studentService.listAllStudents();
        model.addAttribute("allstudents", list);
        model.addAttribute("success", "Student " + student.getName() + " updated successfully.");
        mv.setViewName("allStudents");
    }
    returnmv;
}

/**
 * This method will Delete a Student by Id
 *
 * @param id
 * @return
 */
@RequestMapping(value = { "/delete-{id}" }, method = RequestMethod.GET)
public ModelAndViewdeleteStudent(@PathVariable Long id, ModelMapmodel)
{
    ModelAndViewmv = newModelAndView();
    Student student = studentService.getStudent(id);
    studentService.delete(id);
    List<Student>list = studentService.listAllStudents();
    model.addAttribute("allstudents", list);
    model.addAttribute("success", "Student " + student.getName() + " deleted successfully.");

```

```
        mv.setViewName("allStudents");  
        return mv;  
    }  
}
```

Step 4: Create the service layer into package com.ignou.mvcapp.service package with the following classes.

StudentService.java

```
package com.ignou.mvcapp.service;  
public interface StudentService  
{  
    Student getStudent(Long id);  
    Long saveStudent(Student st);  
    List<Student> listAllStudents();  
    void update(Long id, Student st);  
    void delete(Long id);  
    boolean isStudentUnique(Long id);  
}
```

StudentServiceImpl.java

```
package com.ignou.mvcapp.service;  
@Service("studentService")  
public class StudentServiceImpl implements StudentService  
{  
    @Autowired  
    private StudentDao studentDao;  
  
    @Override  
    public Student getStudent(Long id)  
    {  
        return studentDao.getStudent(id);  
    }  
  
    @Override  
    public Long saveStudent(Student st)  
    {  
        return studentDao.saveStudent(st);  
    }  
  
    @Override  
    public List<Student> listAllStudents()  
    {  
        return studentDao.listAllStudents();  
    }  
  
    @Override  
    public void update(Long id, Student st)  
    {  
        Student stEntity = studentDao.getStudent(id);  
        if (stEntity != null)  
        {  
            stEntity.setFirstName(st.getFirstName());  
            stEntity.setLastName(st.getLastName());  
            stEntity.setGrade(st.getGrade());  
            studentDao.updateStudent(stEntity);  
        }  
    }  
  
    @Override  
    public void delete(Long id)  
    {  
        Student stEntity = studentDao.getStudent(id);  
        if (stEntity != null)  
        {  
            studentDao.deleteStudent(stEntity);  
        }  
    }  
}
```



```

@Override
public boolean isStudentUnique(Long id)
{
    Student student = studentDao.getStudent(id);
    return (student == null || (id != null & !id.equals(student.getId())));
}
}

```

Step 5: Create the model into package **com.ignou.mvcapp.model**.

```

package com.ignou.mvcapp.model;
@Entity
@Table(name = "students")
public class Student
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "grade", nullable = false)
    private Integer grade;

    public Long getId()
    {
        return id;
    }

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

    public String getFirstName()
    {
        return firstName;
    }

    public void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }

    public String getLastName()
    {
        return lastName;
    }

    public void setLastName(String lastName)
    {
        this.lastName = lastName;
    }

    public Integer getGrade()
    {
        return grade;
    }

    public void setGrade(Integer grade)
    {
        this.grade = grade;
    }

    public String getName()
    {
        return this.firstName + " " + this.lastName;
    }
}

```

ignou
THE PEOPLE'S
UNIVERSITY

Step 6: Create the DAO layer into the package `com.ignou.mvcapp.dao`. DAO layer is responsible for interacting with the database.

StudentDao.java

```
package com.ignou.mvcapp.model;

public interface StudentDao
{
    Student getStudent(Long id);

    Long saveStudent(Student st);

    List<Student> listAllStudents();

    void updateStudent(Student st);

    void deleteStudent(Student st);
}
```

StudentDaoImpl.java

```
package com.ignou.mvcapp.model;

@Repository("studentDao")
@Transactional
public class StudentDaoImpl implements StudentDao
{
    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public Student getStudent(Long id)
    {
        Session session = sessionFactory.getCurrentSession();
        Student s = session.get(Student.class, id);
        return s;
    }

    @Override
    public Long saveStudent(Student st)
    {
        Session session = sessionFactory.getCurrentSession();
        session.save(st);
        return st.getId();
    }

    @Override
    public List<Student> listAllStudents()
    {
        Session session = sessionFactory.getCurrentSession();
        CriteriaBuilder cb = session.getCriteriaBuilder();
        CriteriaQuery<Student> cq = cb.createQuery(Student.class);
        Root<Student> root = cq.from(Student.class);
        cq.select(root);
        Query query = session.createQuery(cq);
        @SuppressWarnings("unchecked")
        List<Student> students = query.getResultList();
        return students;
    }

    @Override
    public void updateStudent(Student st)
    {
        Session session = sessionFactory.getCurrentSession();
        session.update(st);
    }

    @Override
    public void deleteStudent(Student st)
    {
    }
}
```

```

{
    Session session = sessionFactory.getCurrentSession();
    session.delete(st);
}
}

```

Step 7: Create the hibernate properties into src/main/resources/application.properties file. Change the username and password for the Mysql installed into your environment.

```

jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/schoolDB
jdbc.username = root
jdbc.password = root

hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = true

hibernate.hbm2ddl.auto = update

```

Step 8: Create the views (JSP) inside src/main/webapp/WEB-INF/views.

addStudentForm.jsp

```

<%@page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add New Student Form</title>

<link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
    rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>
<style type="text/css">
<
    style.error
    {
        color: #ff0000;
    }
</style>

</head>

<body>
    <nav class="navbar navbar-light bg-light">
        <div class="container">
            <div class="navbar-brand mx-auto text-success" href="#">
                <h1>Student Management System (SMS)</h1>
            </div>
        </div>
    </nav>
    <div class="container">
        <div class="text-secondary text-center">
            <h2>Student Form</h2>
        </div>

        <c:if test="${not empty error}">
            <div class="alert alert-danger alert-dismissible fade show"
                role="alert">${error}
                <button type="button" class="btn-close" data-bs-dismiss="alert"
                    aria-label="Close"></button>
            </div>
        </c:if>
        <hr/>
        <form:form method="POST" modelAttribute="student"

```

```

        cssClass="form-horizontal">
        <form:input type="hidden" path="id" id="id"/>

        <div class="form-group">
            <label for="firstName">First Name</label>
        <form:input path="firstName" id="firstName" cssClass="form-control"/>
        </div>

        <div class="form-group">
            <label for="lastName">Last Name</label>
        <form:input path="lastName" id="lastName" cssClass="form-control"/>
        </div>

        <div class="form-group">
            <label for="grade">Grade</label>
        <form:input path="grade" id="grade" cssClass="form-control"/>
        </div>
        <hr/>
        <c:choose>
            <c:when test="${edit}">
                <button type="submit" class="btn btn-
primary">Update</button>
            </c:when>
            <c:otherwise>
                <button type="submit" class="btn btn-
primary">Save</button>
            </c:otherwise>
        </c:choose>

        <a class="btn btn-secondary" href="{c:url value='/'}/>List
of All Students</a>
    </form:form>
</div>
</body>
</html>

```

allStudents.jsp

```

<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>All Students</title>

<link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
    rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>
<link href="{c:url value='/css/custom.css'}/>" rel="stylesheet">

</head>
<body>
    <nav class="navbar navbar-light bg-light">
        <div class="container">
            <div class="navbar-brand mx-auto text-success" href="#">
                <h1>Student Management System (SMS)</h1>
            </div>
        </div>
    </nav>
    <div class="container">
        <div class="text-secondary text-center">
            <h2>Registered Students</h2>
        </div>

        <c:if test="${not empty success}">
            <div class="alert alert-success alert-dismissible fade show"
                role="alert">${success}
                <button type="button" class="btn-close" data-bs-
dismiss="alert"
                aria-label="Close"></button>
            </div>
        </c:if>
    </div>

```

```

        </div>
    </c:if>

    <hr/>
    <table class="table">
        <thead class="thead-dark">
            <tr>
                <td>Student ID</td>
                <td>Student Name</td>
                <td>Grade</td>
                <td>Modify</td>
                <td>Delete</td>
            </tr>
        </thead>
        <c:forEach items="${allstudents}" var="student">
            <tr>
                <td>${student.id}</td>
                <td>${student.firstName}&nbsp;${student.lastName}</td>
                <td>${student.grade}</td>
                <td><a href="<c:urlvalue='/edit-${student.id}'/">" class="btn btn-success">Modify</a></td>
                <td><a href="<c:urlvalue='/delete-${student.id}'/">" class="btn btn-danger">Delete</a></td>
            </tr>
        </c:forEach>
    </table>
    <hr/>
    <div class="form-group">
        <a class="btn btn-primary" href="<c:urlvalue='/new'/">">Add
        New Student</a>
    </div>
</div>
</body>
</html>

```

Step 9: Create the custom css inside folder src/main/webapp/resources/static/css.

custom.css

```

.btn{font-size:10px }
tr:first-child{font-size:20px; color:maroon; font-weight:bold;}

```

Step 10: Build the project using maven with goal clean install and then deploy the generated war file inside the target directory to the external tomcat. You can directly execute the project into the eclipse by using the Run on Server option. Once the application is up and running, access the URL <http://localhost:8080/spring-mvc-crud-app/> and perform the CRUD operation provided by the application.

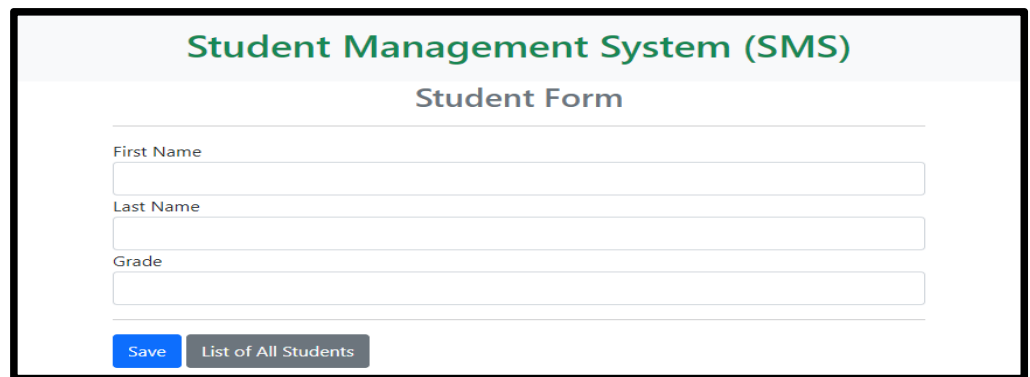
Output:

Home screen renders all existing students by executing read operation into the database.

Student Management System (SMS)				
Registered Students				
Student ID	Student Name	Grade	Modify	Delete
2	Prasoon Singh	9	Modify	Delete
3	Swati Rathor	10	Modify	Delete
6	Anurag	10	Modify	Delete
Add New Student				

Figure 8.4: CRUD Application Home Screen

Click on the *Add New Student* button, and a form will be opened to create a new student record. Fill the form and click the save button. This operation performs create operation into the database, and a new student record will be created.



Student Management System (SMS)

Student Form

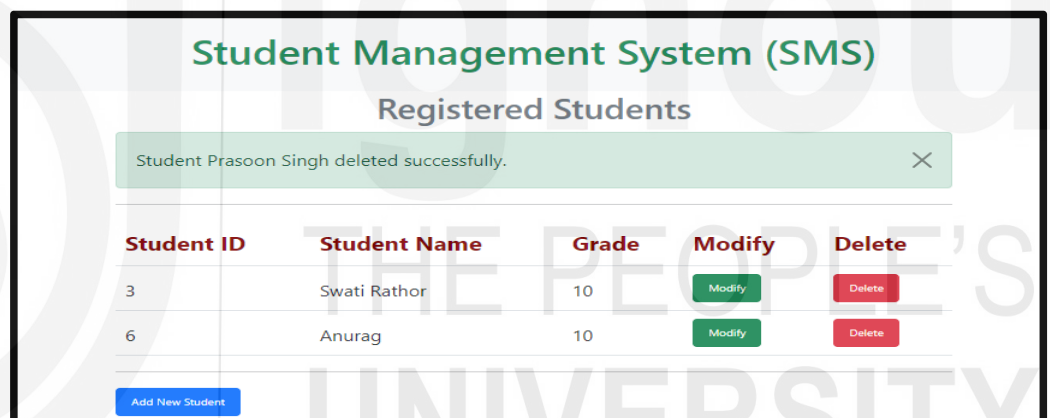
First Name

Last Name

Grade

Figure 8.5: CRUD Application Create Form

A student record can be either modified or deleted from the home screen. The click of modifying button will render the student record in editable mode. Any attribute can be updated. This operation will perform an update operation into the database. While on click of the delete button will delete the student record from the database.



Student Management System (SMS)

Registered Students

Student Prasoon Singh deleted successfully. ✕

Student ID	Student Name	Grade	Modify	Delete
3	Swati Rathor	10	<input type="button" value="Modify"/>	<input type="button" value="Delete"/>
6	Anurag	10	<input type="button" value="Modify"/>	<input type="button" value="Delete"/>

Figure 8.6: CRUD Application Delete Screen

Check Your Progress 3

- 1) Define CRUD within computer programming.

.....

.....

.....

.....

- 2) Define the mapping of the CRUD operations with SQL statements with suitable examples.

.....

.....

- 3) Define the mapping of CRUD in the Rest context.

8.7 SUMMARY

The unit has briefly explained the Bootstrap CSS framework. Bootstrap is a feasible solution to tackle the challenges of making a responsive website as it evolves more and more. The Bootstrap makes things a lot easier to make the website responsive. Bootstrap CSS framework configuration into Spring MVC web application has been described in detail. Custom CSS can be used to override the Bootstrap styles in order to customize the Bootstrap and match the theme of a website. Hibernate is a very popular ORM framework to interact with databases, and Spring provides the required implementation to integrate Hibernate and Spring. The highlights of this unit are as follows.

- Bootstrap is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development.
- There are two ways to include Bootstrap into JSP or HTML.
 - Bootstrap through Content Delivery Network (CDN)
 - Including by downloading files
- CSS allows you to customize the style of your website and apply stylistic decisions across its entirety.
- CSS allows you to separate the style from the structure of a web page.
- There are three main components for adding customized CSS language to the website. The components are **selectors, properties and values**.
- Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate.
- Spring provides two key beans in order to support the hibernate integration.
 - **LocalSessionFactoryBean**
 - **PlatformTransactionManager**
- Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE.
- In SQL context, CRUD corresponds to Insert, Select, Update and Delete, respectively.
- In REST context, CRUD corresponds to Rest web service POST, GET, PUT and DELETE, respectively.

8.7 SOLUTIONS/ ANSWER TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) Bootstrap is a free, popular and open-source CSS framework directed at responsive, mobile-first front-end web development. Unlike many web frameworks, it concerns itself with front-end development only. It contains HTML and CSS based design templates for various interface

components such as typography, forms, buttons, modals, image carousels, navigation etc, as well as optional JavaScript extensions. Bootstrap provides the ability to create responsive designs faster and easier. The various factors which make Bootstrap a popular choice for CSS framework are as follows.

- (i) All popular web browser support
 - (ii) Responsive design
 - (iii) Uniform Solution to build an interface
 - (iv) Functional built-in components
- 2) The Bootstrap 5 is the most recent version of the Bootstrap framework. There are two ways to include Bootstrap into JSP or HTML.
- (i) **Bootstrap through Content Delivery Network (CDN):** Bootstrap CDN is an efficient and faster way to deliver the content from your website to the users. Bootstrap CDN speeds up the websites as CDN servers are intelligent enough to serve the resources based on proximity.
 - (ii) **Offline or Downloading files locally:** Another approach to include Bootstrap is to directly download the Bootstrap CSS and JS files locally to the project folder and then include the local copy into JSP/HTML.
- 3) For the development environment, offline usage of Bootstrap is suitable. For the production environment, CDN is better, and the following obvious reasons support to prefer CDN configuration for Bootstrap-
- o CDN servers are intelligent enough to serve the resources based on proximity
 - o They are super-fast at serving the content
 - o The CDN network is spread across the world.
 - o If you think that your server will challenge the above, then go for it

Check Your Progress 2

- 1) CSS is an extension of HTML that outlines specific stylistic instructions to style websites. CSS allows you to customize the look of your website and apply stylistic decisions across its entirety. CSS allows you to separate the style from the structure of a web page. CSS is used to specify the color of a heading or what font your content should be written in. To keep the style uniformly across the pages, a separate CSS file with any name e.g. custom.css can be created and custom styles can be added to this file. The custom CSS can be included at the top of each webpage inside the <head> tag using the below instruction.

```
<link rel="stylesheet" type="text/css" href="css/custom.css">
```

- 2) The below style instructions change all of the text within <p> tags on a page to red.

```
<head>
<style type="text/css">
p
{
    color: red;
}
</style>
</head>
<body>
<p>
This is Red Color Text.
```



```
</p>
<p>
This is another paragraph with red color.
</p>
</body>
```

- 3) Spring supports bootstrapping the Hibernate SessionFactory in order to set up the database using Hibernate. The database setup can be done with a few lines of Java code or XML configuration. Spring provides two key beans, in order to support the hibernate integration, available in the org.springframework.orm.hibernate5 package:
 - **LocalSessionFactoryBean:** creates a Hibernate's **SessionFactory**, which is injected into Hibernate-based DAO classes.
 - **PlatformTransactionManager:** provides transaction support code for a **SessionFactory**. Programmers can use **@Transactional** annotation in DAO methods to avoid writing boiler-plate transaction code explicitly.
- 4) The Java configuration to integrate Hibernate with Spring is as follows.

```
package com.ignou.mvcapp;

@Configuration
@EnableTransactionManagement
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfig
{
    @Autowired
    private Environment environment;

    @Bean
    public LocalSessionFactoryBean sessionFactory()
    {
        LocalSessionFactoryBean sessionFactory =
        new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan(new String[] {
        "com.ignou.mvcapp.model" });
        sessionFactory.setHibernateProperties(hibernateProperties());
        return sessionFactory;
    }

    @Bean
    public DataSource dataSource()
    {
        DriverManagerDataSource dataSource =
        new DriverManagerDataSource();
        // set datasource properties
        return dataSource;
    }

    private Properties hibernateProperties()
    {
        Properties properties = new Properties();
        //set Hibernate properties
        return properties;
    }
}
```

```
@Bean
public PlatformTransactionManager getTransactionManager()
{
    HibernateTransactionManager htm =
new HibernateTransactionManager();
    htm.setSessionFactory(sessionFactory().getObject());
    return htm;
}
}
```

Check Your Progress 3

- 1) Within computer programming, the acronym CRUD stands for CREATE, READ, UPDATE and DELETE. The CRUD paradigm is common in constructing web applications. While constructing the APIs, the model should provide four basic types of functionality such as Create, Read, Update and Delete the resources. CRUD operations are also often used with SQL.

It can also describe user-interface conventions that allow viewing, searching and modifying information through computer-based forms and reports. Most applications have some form of CRUD functionality. When programmers provide interactions with this database (often through stored procedures), the steps of CRUD, for a Student Management System, are carried out as follows:

- o Create: A new student is entered into the database.
- o Read: The student's information is displayed to the users.
- o Update: Already existing student's attributes are being updated with new values.
- o Delete: If the student is not part of the institute, the student can be deleted from the records.

- 2) CRUD to database SQL operation mapping with examples are as follows.

Function	SQL Statement	Example
C(reate)	Insert	Insert into student (name,grade) values('Prasoon',10);
R(ead)	Select	Select * from student
U(pdate)	Update	Update student set grade=10 where id=1;
D(elete)	Delete	Delete from student where id=1;

- 3) In REST context, CRUD corresponds to Rest web service POST, GET, PUT and DELETE respectively. The following table maps CRUD to REST with an example.

Function	Rest web service	Example
C(reate)	POST	POST http://teststudent.com/students/
R(ead)	GET	GET http://teststudent.com/students/
U(pdate)	PUT	PUT http://teststudent.com/students/1
D(elete)	DELETE	DELETE http://teststudent.com/students/1

8.9 REFERENCES/FURTHER READING

- ... Craig Walls, "Spring Boot in action" Manning Publications, 2016.
(<https://doc.lagout.org/programming/Spring%20Boot%20in%20Action.pdf>)

- ... Christian Bauer, Gavin King, and Gary Gregory, “Java Persistence with Hibernate”,Manning Publications, 2015.
- ... Ethan Marcotte, “Responsive Web Design”, Jeffrey Zeldman Publication, 2011.(http://nadin.miem.edu.ru/images_2015/responsive-web-design-2nd-edition.pdf)
- ... Tomcy John, “Hands-On Spring Security 5 for Reactive Applications”,Packt Publishing,2018.
- ... <https://www.creative-tim.com/blog/web-design/add-bootstrap-html-guide/>
- ... <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
- ... <https://getbootstrap.com/docs/5.0/examples/>
- ... <https://wordpress.com/go/web-design/what-is-css/>
- ... <https://getbootstrap.com/docs/5.0/customize/overview/>
- ... <https://www.baeldung.com/hibernate-5-spring>
- ... <https://stackoverflow.com/questions/22693452/bootstrap-with-spring-mvc/22765275>
- ... <https://www.quora.com/Is-it-better-to-use-Bootstrap-offline-or-just-include-a-link-to-the-CDN>

