Spring MVC Integration with MySQL

Last Updated : 23 Jul, 2025

Spring MVC is one of the most popular Java frameworks for building scalable web applications. When combined with MySQL, it provides a robust solution for developing data-driven applications. This article will guide you through integrating **Spring MVC with MySQL, covering database setup, project configuration, DAO layer implementation, and CRUD operations.** Whether you are a beginner or an experienced developer, this step-by-step article will help you build a Spring MVC MySQL application efficiently.

Steps Involved in the Creation of Database and Table

Step 1: Create the Database

First, we will create a database **test**.

Note: test is the name of the database here.

```
CREATE DATABASE test;
```

Step 2: Make the Database test an Active

```
use test;
```

Step 3: Create the Table

Create a table named **studentsdetails** with the following structure

```
CREATE TABLE studentsdetails (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(25),
    caste VARCHAR(25),
    neetmarks INT,
    gender VARCHAR(10)
);
```

Step 4: Insert Records into the Table

```
INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek1', 'OBC', 600, 'Female');

INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek2', 'General', 700, 'Female');

INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek3', 'General', 600, 'Male');

INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek4', 'OBC', 670, 'Male');

INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek5', 'SC', 600, 'Female');

INSERT INTO studentsdetails(name, caste, neetmarks, gender)
VALUES ('Geek6', 'SC', 500, 'Male');
```

Step 5: Select Data from the Table

Retrieve all records from the studentsdetails table

```
SELECT * FROM studentsdetails;
```

The below image demonstrates a database table showing student data, including their names, caste, NEET marks and gender

Result Grid | Filter Rows:

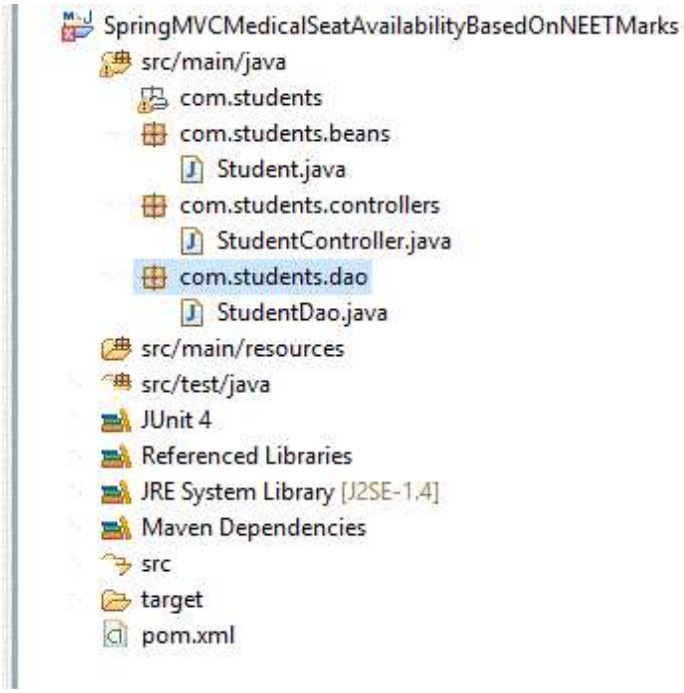
	id	name	caste	neetmarks	gender
▶	1	Geek1	OBC	600	Female
	2	Geek2	General	700	Female
	3	Geek3	General	600	Male
	4	Geek4	OBC	670	Male
	5	Geek5	SC	600	Female
	6	Geek6	SC	500	Male
*	NULL	NULL	NULL	NULL	NULL

Spring MVC Application Setup

Now let us do all the necessary steps in the Spring MVC application.

Project Structure:

The **project structure** for the **Spring MVC application** is as follows:



Step 1: pom.xml Configuration

The pom.xml file includes the necessary dependencies for Spring MVC, MySQL, and other libraries.

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
```

```
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.students</groupId>

<artifactId>SpringMVCMedicalSeatAvailabilityBasedOnNEETMarks</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Spring MVC Dependency -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.18</version>
        </dependency>

        <!-- MySQL Connector Dependency -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
        </dependency>

        <!-- Spring JDBC Dependency -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.3.18</version>
        </dependency>

        <!-- JUnit Dependency -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>

<finalName>SpringMVCMedicalSeatAvailabilityBasedOnNEETMarks</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
```

```
        </plugins>
    </build>
</project>
```

Step 2: spring-servlet.xml

The spring-servlet.xml file is used to configure the Spring MVC application context and database connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans/"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context/"
    xmlns:mvc="http://www.springframework.org/schema/mvc/"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans/
        http://www.springframework.org/schema/beans//spring-beans.xsd
        http://www.springframework.org/schema/context/
        http://www.springframework.org/schema/context//spring-context.xsd
        http://www.springframework.org/schema/mvc/
        http://www.springframework.org/schema/mvc//spring-mvc.xsd">

    <context:component-scan base-package="com.students.controllers" />

    <!-- Database Configuration -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/test?
useSSL=false&amp;serverTimezone=UTC" />
        <property name="username" value="root" />
        <property name="password" value="yourpassword" /> <!-- Add your
MySQL password here -->
    </bean>

    <!-- JdbcTemplate Configuration -->
    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- DAO Configuration -->
    <bean id="studentDao" class="com.students.dao.StudentDao">
        <property name="template" ref="jdbcTemplate" />
    </bean>
</beans>
```

Let us now roll on over to '**bean** class.' The fields in this bean class should be equivalent to the **MySQL table** structure. Then only it will be easier and more effective to communicate.

Step 3: Student.java (Bean Class)

The Student class maps to the studentsdetails table.

```
// Java Program to Illustrate Student Class

// Class
public class Student {

    // Class data members

    // Map to studentsdetails.id
    private int id;
    // Map to studentsdetails.name
    private String name;
    // Map to studentsdetails.caste
    private String caste;
    // Map to studentsdetails.neetMarks
    private int neetMarks;
    // Map to studentsdetails.gender
    private String gender;

    // Getter and setter methods

    // Getter
    public int getNeetMarks() { return neetMarks; }

    // Setter
    public void setNeetMarks(int neetMarks)
    {
        this.neetMarks = neetMarks;
    }

    // Getter
    public String getGender() { return gender; }

    // Setter
    public void setGender(String gender)
    {
        this.gender = gender;
    }

    // Getter
    public int getId() { return id; }
```

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

    // Getter
    public String getName() { return name; }

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

    // Getter
    public String getCaste() { return caste; }

    // Setter
    public void setCaste(String caste)
    {
        this.caste = caste;
    }
}
```

Now to do the database operations, we need the DAO java file

Step 4: StudentDao.java (DAO Class)

The StudentDao class handles database operations.

```
// Java Program to Illustrate StudentDao Class

// Importing required classes
import com.students.beans.Student;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;

// Class
public class StudentDao {

    private JdbcTemplate template;

    public void setTemplate(JdbcTemplate template) {
        this.template = template;
    }

    // Method to search student by name
    public Student getStudentByName(String studentName) {
        String sql = "SELECT * FROM studentsdetails WHERE name=?";
        try {
            return template.queryForObject(
                sql, new Object[] { studentName },
                new BeanPropertyRowMapper<>(Student.class));
        } catch (Exception e) {
            // Return null if no student is found
```

```
            return null;
        }
    }

    // Method to search student by caste
    public Student getStudentByCaste(String caste) {
        String sql = "SELECT * FROM studentsdetails WHERE caste=?";
        try {
            return template.queryForObject(
                sql, new Object[] { caste },
                new BeanPropertyRowMapper<>(Student.class));
        } catch (Exception e) {
            // Return null if no student is found
            return null;
        }
    }

    // Method to search student by ID
    public Student getStudentById(int id) {
        String sql = "SELECT * FROM studentsdetails WHERE id=?";
        try {
            return template.queryForObject(
                sql, new Object[] { id },
                new BeanPropertyRowMapper<>(Student.class));
        } catch (Exception e) {
            // Return null if no student is found
            return null;
        }
    }

    // Method to search student by NEET marks
    public Student getStudentByNeetMarks(int neetMarks) {
        String sql = "SELECT * FROM studentsdetails WHERE neetMarks=?";
        try {
            return template.queryForObject(
                sql, new Object[] { neetMarks },
                new BeanPropertyRowMapper<>(Student.class));
        } catch (Exception e) {
            // Return null if no student is found
            return null;
        }
    }

    // Additional business logic can be added here
}
```

Let us see the controller class now

Step 5: StudentController.java

The StudentController class handles HTTP requests.

```java
// Java Program to Illustrate StudentController Class

// Importing required classes
import com.students.beans.Student;
import com.students.dao.StudentDao;
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 org.springframework.web.servlet.ModelAndView;

// Class
@Controller
public class StudentController {

    // Autowired to inject StudentDao
    private final StudentDao dao;

    @Autowired
    public StudentController(StudentDao dao) {
        this.dao = dao;
    }

    // Method to display the search form
    @RequestMapping("/studentsearchform")
    public String showSearchForm(Model model) {
        model.addAttribute("command", new Student());
        return "studentsearchform";
    }

    // Method to check medical seat eligibility based on NEET marks
    @RequestMapping(value = "/checkByNeetMarks", method =
RequestMethod.POST)
    public ModelAndView checkByNeetMarks(@ModelAttribute("student") Student
student) {
        ModelAndView mav = new ModelAndView("welcome");

        // Fetch student details from the database
        Student studentData = dao.getStudentByName(student.getName());

        if (studentData != null) {
            // Determine eligibility based on caste and NEET marks
            boolean isEligible = false;
            String caste = studentData.getCaste();
            int neetMarks = studentData.getNeetMarks();

            if (caste.equalsIgnoreCase("General") && neetMarks >= 600) {
                isEligible = true;
            } else if (caste.equalsIgnoreCase("OBC") && neetMarks >= 500) {
                isEligible = true;
            } else if (caste.equalsIgnoreCase("SC") && neetMarks >= 400) {
                isEligible = true;
            }
```

```
            // Add data to the ModelAndView object
            mav.addObject("firstname", studentData.getName());
            mav.addObject("caste", caste);
            mav.addObject("neetmarks", neetMarks);
            mav.addObject("availability", isEligible ? "Eligible to get
```



```
            // If student is not found in the database
            mav.addObject("firstname", student.getName());
            mav.addObject("availability", "Student not found in the
database");
        }

        return mav;
    }
}
```

Step 6: Deployment and Testing

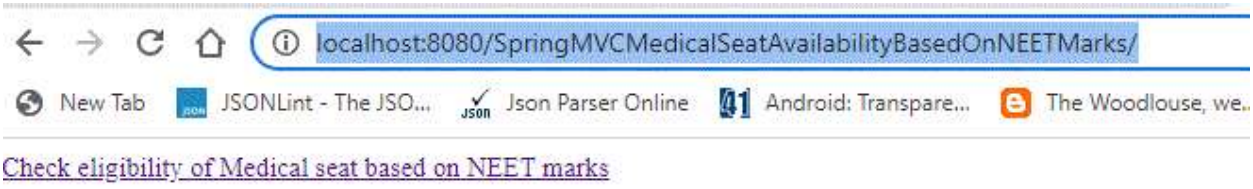
Deploy the application as a WAR file under the Tomcat webapps folder.

At the start of Tomcat, the above application can be invoked by using.

Start Tomcat and access the application using

> *http://localhost:8080/SpringMVCMedicalSeatAvailabilityBasedOnNEETMarks/*

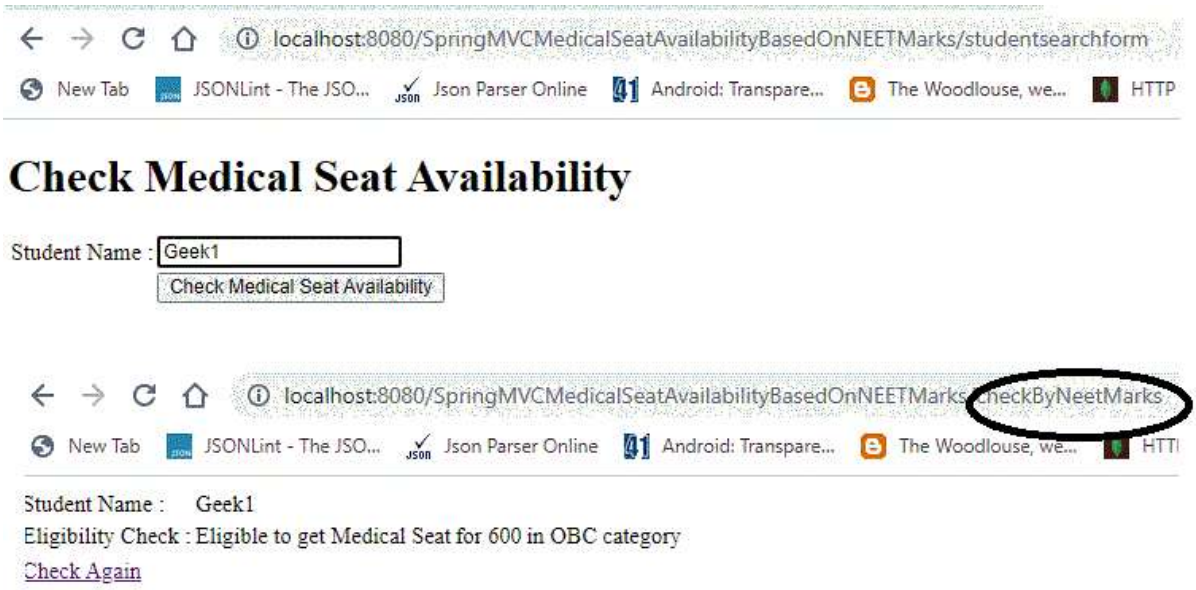
Use the form to check medical seat eligibility based on NEET marks.



After clicking the link, we will get output as below:

Check for the url change. All should be aligned with controller

Usecase:



According to the logic written, we are getting results, here:

- Geek1 is the name given for the search. It will be checked against the "studentsdetails" table
- Circled one indicates the name of the request mapping.

It is a sample application and the necessary steps are given in this which interacts with the MySQL database. Using Spring MVC and MySQL, we can easily carry out the business logic easily.