Assignment No:8

Title: Design and develop a website using Springboot.

Problem Statement: Design and develop a responsive website to prepare one semester result of VIT students using REACT, Springboot and MySQL/ MongoDB/Oracle. Take any four subjects with MSE Marks (30%) ESE Marks (70%).

Course Objective: To learn REST API based enterprise website development using REACT, Node JS, Spring Boot with different database technologies.

Course Outcome: Write Web API/RESTful API application programming interface to communicate with Springboot as a server side technology.

Tools Required: STS, MySql, Firefox/GoogleChrome

Theory:

1. Spring Boot

Spring Boot is a Java framework built on top of Spring that simplifies application development. It eliminates boilerplate code with auto-configuration. Spring Boot comes with an embedded server, making applications production ready out of the box. It supports web apps, REST APIs, microservices, security and seamless cloud deployment.

1.1 Why Spring Boot?

- Fast Development: Quickly build production ready apps with minimal configuration.
- Auto-Configuration: Reduces boilerplate code, you can focus on business logic.
- Microservices Friendly: Perfect for building and managing distributed systems.
- Seamless Integrations: Works easily with databases, security, messaging and cloud platforms.
- Wide Adoption: Backed by Spring ecosystem and used in real world enterprise applications.

1.2 Applications of Spring Boot

Following is the list of few of the great benefits of using Spring Boot Framework −

- **POJO Based** - Spring Boot enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- **Modular** - Spring Boot is modular by nature. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
- **Integration with existing frameworks** - Spring Boot does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM

frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.

- **Testablity** - Testing an application written with Spring Boot is very easy because environment-dependent code is moved into this framework. Furthermore, by using Java Bean style POJOs, it becomes easier to use dependency injection for injecting test data.
- **Web MVC** - Spring Boot's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
- **Central Exception Handling** - Spring Boot provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
- **Lightweight** - Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- **Transaction management** - Spring Boot provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

2. Simple Project in STS

2.1 Project Structure

```
HelloWorldSpringBoot/
 │
 ├── src/main/java/com/example/helloworld/
 │    ├── HelloWorldApplication.java
 │    └── HelloController.java
 │
 ├── src/main/resources/
 │    ├── application.properties
 │
 ├── pom.xml
```

2.2 HelloWorldApplication.java

(Location: src/main/java/com/example/helloworld/HelloWorldApplication.java)

```
package com.example.helloworld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class HelloWorldApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloWorldApplication.class, args);
        System.out.println("🚀 Spring Boot Hello World Application Started!");
    }
}
```

2.3 HelloController.java
(Location: src/main/java/com/example/helloworld/HelloController.java)

```
package com.example.helloworld;

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

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World from Spring Boot!";
    }
}
```


2.4 application.properties
(Location: src/main/resources/application.properties)

```
server.port=8080
spring.application.name=HelloWorldSpringBoot
```

2.5 Run the Application

1. **Right-click HelloWorldApplication.java → Run As → Spring Boot App.**
2. **Console will show:**

```
🚀 Spring Boot Hello World Application Started!
Tomcat started on port(s): 8080
```

3. **Open browser: http://localhost:8080/hello**

3. File Details

3. 1. pom.xml (HelloWorldSpringBoot/pom.xml)

- Defines the project as a Maven project.
- Lists dependencies like **spring-boot-starter-web**.
- Manages build plugins and Spring Boot version.

---

3. 2. HelloWorldApplication.java
(src/main/java/com/example/helloworld/HelloWorldApplication.java)

- The **main entry point** of the application.
- Annotated with @SpringBootApplication.
- Starts the Spring Boot application (embedded Tomcat server).

3.3. HelloController.java
(src/main/java/com/example/helloworld/HelloController.java)

- A REST controller (@RestController) that handles HTTP requests.
- Example:
  - URL /hello → returns **"Hello World from Spring Boot!"**.

3.4. application.properties
(src/main/resources/application.properties)

- Stores configuration settings.
- Example:
  - Change port → server.port=8081.
  - Set application name → spring.application.name=HelloApp.

3.5. static/ (Optional)
(src/main/resources/static/)

- Stores static assets like **CSS, JS, Images**.
- Example: src/main/resources/static/style.css → accessible via http://localhost:8080/style.css.

3.6. templates/ (Optional)
(src/main/resources/templates/)

- Stores **HTML pages** (usually with Thymeleaf or JSP).
- Example: index.html here can be returned by a controller.

3.7. HelloWorldApplicationTests.java
(src/test/java/com/example/helloworld/HelloWorldApplicationTests.java)

- Auto-created JUnit test class.
- Used to test if the Spring context loads properly.

4. Spring Boot Annotations

Annotation	Meaning / Usage
1. @SpringBootApplication	Main entry point of a Spring Boot app. Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.

Annotation	Meaning / Usage
2. @RestController	Marks a class as a REST API controller (@Controller + @ResponseBody).
3. @Controller	Handles web requests (usually returns HTML views).
4. @GetMapping	Maps **HTTP GET** requests to a method.
5. @PostMapping	Maps **HTTP POST** requests to a method.
6. @PutMapping	Maps **HTTP PUT** requests to a method.
7. @DeleteMapping	Maps **HTTP DELETE** requests to a method.
8. @RequestMapping	General mapping for any HTTP request (GET/POST/PUT/DELETE).
9. @RequestParam	Extracts values from query parameters (e.g., /search?name=Tom).
10. @PathVariable	Extracts values from the URL path (e.g., /user/{id}).
11. @RequestBody	Maps JSON/XML request body to a Java object.
12. @ResponseBody	Sends return value of a method directly as HTTP response.
13. @Autowired	Injects (wires) dependencies automatically.
14. @Service	Marks a class as a **service layer bean** (business logic).
15. @Repository	Marks a class as a **DAO/repository bean** (database operations).
16. @Component	Generic annotation for Spring-managed beans.
17. @Entity	Defines a JPA entity (maps class to database table).
18. @Id	Marks a field as **primary key** in a JPA entity.
19. @GeneratedValue	Defines how the primary key is generated (AUTO, IDENTITY, SEQUENCE).
20. @Column	Maps a field to a database table column.

5. SQL (Structured Query Language)

Sql is a standard language used to manage and interact with databases. It allows you to store, retrieve, update, and delete data in a relational database.

MySQL is the most popular Open Source Relational SQL database management system. MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is popular because:

- It is released under open source
- It handles a large subset of the functionality of the most expensive and powerful database packages, it uses standard form of the well-known SQL data language,
- works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets and with PHP. MySQL supports large databases, up to 50 million rows or more in a table.
- The default file size limit for a table is 4GB, but we can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB)

5.1 Basic MySql Query

5.1.1 MySQL Connection

```
[root@host]# mysql -u root -p
Enter password:******
```

5.1.2 Create Database

mysql> **CREATE DATABASE menagerie;**

5.1.3 Selecting Database

mysql> **USE menagerie**
Database changed

5.1.4 Drop Database

mysql> DROP DATABASE TUTORIALS

5.1.5 Create MySQL Tables

Syntax:
CREATE TABLE *table_name*
(
column_name1 data_type(*size*),
column_name2 data_type(*size*),
....
);

Example:
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
);

5.1.6 ALTER

Syntax:
ALTER TABLE table_name ADD column_name datatype;
ALTER TABLE table_name DROP COLUMN column_name;
ALTER TABLE table_name MODIFY COLUMN column_name datatype;

Example:
ALTER TABLE Persons ADD DateOfBirth date;
ALTER TABLE Persons ALTER COLUMN DateOfBirth int;
ALTER TABLE Persons DROP COLUMN DateOfBirth;

5.1.7 DROP

DROP TABLE table_name;
DROP DATABASE database_name;

Conclusion: Hence, we have successfully design and develop a responsive website to prepare one semester result of VIT students using Springboot and MySQL.

Source Code

Step 1: Create the Spring Boot project (STS wizard)

1. In STS: **File → New → Spring Starter Project**.
2. Fill:
    - **Name:** ty-application
    - **Group:** com.example
    - **Artifact:** ty-application (or marks-certificate)
    - **Package:** com.example.demo
    - **Type:** Maven, Packaging: Jar, Java: 17 (or your version)
3. Click **Next** → in Dependencies choose:
    - **Spring Web**
    - **Thymeleaf**
    - **Spring Data JPA**
    - **MySQL Driver**
    - (Optional) Spring Boot DevTools
4. Click **Finish**.

Step 2: Create Java package structure

You need packages for model, controller, repository, and the main app.

1. Expand src/main/java.
2. Create new package com.example.demo.model, com.example.demo.controller, com.example.demo.repository.

Step 3: Create Java files (classes)

1. src/main/java/com/example/demo/TyApplication.java
2. src/main/java/com/example/demo/model/StudentResult.java
3. src/main/java/com/example/demo/repository/StudentResultRepository.java
4. src/main/java/com/example/demo/controller/ResultController.java

Step 4: Create Thymeleaf templates (HTML)

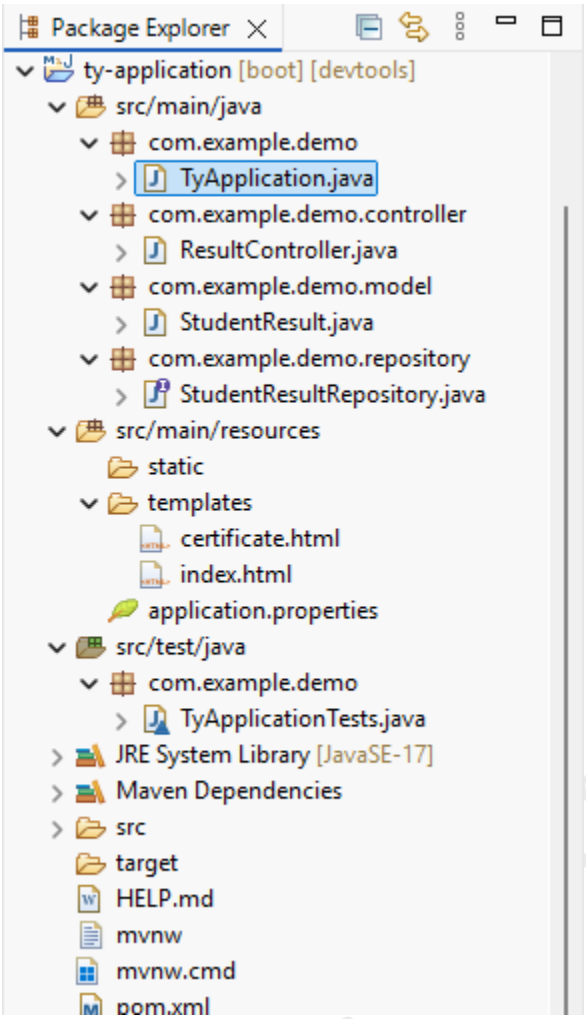
Put these files in src/main/resources/templates/:

- index.html — the marks input form.
- certificate.html — the generated certificate page.

Step 5: Create MySQL database

CREATE DATABASE studentdb;

Project structure (files & locations)



File and Source Code

pom.xml

```
<!-- pom.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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>ty-application</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```
<name>ty-application</name>
<description>Spring Boot App - CGPA calculator and certificate</description>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.2.0</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
  <java.version>17</java.version>
</properties>

<dependencies>
  <!-- Web + Thymeleaf -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- JPA + MySQL -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Optional: devtools for hot reload in development -->
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

TY Application.java

```
// src/main/java/com/example/demo/DemoApplication.java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }
}
```

StudentResult.java

```
// src/main/java/com/example/demo/model/StudentResult.java
package com.example.demo.model;

import jakarta.persistence.*;

@Entity
@Table(name = "student_results")
public class StudentResult {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

private String studentName;

// For each subject: insem and endsem
private int sub1Insem;
private int sub1Endsem;

private int sub2Insem;
private int sub2Endsem;

private int sub3Insem;
private int sub3Endsem;

private int sub4Insem;
private int sub4Endsem;

private int sub5Insem;
private int sub5Endsem;

// computed fields
private int totalMarks;    // sum of 5 subjects (each out of 100) => 0..500
private double percentage; // 0..100
private double cgpa;       // percentage/9.5

public StudentResult() {}

// getters and setters (omitted here for brevity) — include all getters/setters below
// You can generate them in STS or IDE.

// --- getters & setters ---
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getStudentName() { return studentName; }
```

```
    public void setStudentName(String studentName) { this.studentName = studentName; }

    public int getSub1Insem() { return sub1Insem; }
    public void setSub1Insem(int sub1Insem) { this.sub1Insem = sub1Insem; }
    public int getSub1Endsem() { return sub1Endsem; }
    public void setSub1Endsem(int sub1Endsem) { this.sub1Endsem = sub1Endsem; }

    public int getSub2Insem() { return sub2Insem; }
    public void setSub2Insem(int sub2Insem) { this.sub2Insem = sub2Insem; }
    public int getSub2Endsem() { return sub2Endsem; }
    public void setSub2Endsem(int sub2Endsem) { this.sub2Endsem = sub2Endsem; }

    public int getSub3Insem() { return sub3Insem; }
    public void setSub3Insem(int sub3Insem) { this.sub3Insem = sub3Insem; }
    public int getSub3Endsem() { return sub3Endsem; }
    public void setSub3Endsem(int sub3Endsem) { this.sub3Endsem = sub3Endsem; }

    public int getSub4Insem() { return sub4Insem; }
    public void setSub4Insem(int sub4Insem) { this.sub4Insem = sub4Insem; }
    public int getSub4Endsem() { return sub4Endsem; }
    public void setSub4Endsem(int sub4Endsem) { this.sub4Endsem = sub4Endsem; }

    public int getSub5Insem() { return sub5Insem; }
    public void setSub5Insem(int sub5Insem) { this.sub5Insem = sub5Insem; }
    public int getSub5Endsem() { return sub5Endsem; }
    public void setSub5Endsem(int sub5Endsem) { this.sub5Endsem = sub5Endsem; }

    public int getTotalMarks() { return totalMarks; }
    public void setTotalMarks(int totalMarks) { this.totalMarks = totalMarks; }

    public double getPercentage() { return percentage; }
    public void setPercentage(double percentage) { this.percentage = percentage; }

    public double getCgpa() { return cgpa; }
    public void setCgpa(double cgpa) { this.cgpa = cgpa; }
}
```

StudentResultRepository.java

```
// src/main/java/com/example/demo/repository/StudentResultRepository.java
```

```
package com.example.demo.repository;

import com.example.demo.model.StudentResult;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentResultRepository extends JpaRepository<StudentResult, Long> {
}
```

ResultController.java

```
// src/main/java/com/example/demo/controller/ResultController.java
package com.example.demo.controller;

import com.example.demo.model.StudentResult;
import com.example.demo.repository.StudentResultRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;


@Controller
public class ResultController {

    @Autowired
    private StudentResultRepository repository;

    @GetMapping("/")
    public String showForm(Model model) {
        model.addAttribute("studentResult", new StudentResult());
        return "index";
    }

    @PostMapping("/calculate")
    public String calculateAndStore(@ModelAttribute StudentResult studentResult, Model
model) {

        // Calculate totals per subject (insem + endsem)
```

```
        int s1 = studentResult.getSub1Insem() + studentResult.getSub1Endsem();
        int s2 = studentResult.getSub2Insem() + studentResult.getSub2Endsem();
        int s3 = studentResult.getSub3Insem() + studentResult.getSub3Endsem();
        int s4 = studentResult.getSub4Insem() + studentResult.getSub4Endsem();
        int s5 = studentResult.getSub5Insem() + studentResult.getSub5Endsem();

        int totalMarks = s1 + s2 + s3 + s4 + s5; // out of 500
        double percentage = (totalMarks / 5.0);   // average percent
        double cgpa = percentage / 9.5;           // convert percent to CGPA (common formula)

        // set computed fields
        studentResult.setTotalMarks(totalMarks);
        studentResult.setPercentage(Math.round(percentage * 100.0) / 100.0);
        studentResult.setCgpa(Math.round(cgpa * 100.0) / 100.0);

        // persist to DB
        StudentResult saved = repository.save(studentResult);

        // pass saved entity to certificate view
        model.addAttribute("result", saved);

        // show certificate page
        return "certificate";
    }
}
```

application.properties

```
# src/main/resources/application.properties

# MySQL datasource - change username,password,port/db as needed
spring.datasource.url=jdbc:mysql://localhost:3306/studentdb?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# JPA / Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true

# Thymeleaf
spring.thymeleaf.cache=false
```

index.html

```
<!-- src/main/resources/templates/index.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8" />
  <title>CGPA Calculator</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body class="bg-light">
<div class="container mt-5">
  <div class="card shadow">
    <div class="card-header">
      <h3>Enter Marks (each subject = Insem (30) + Endsem (70))</h3>
    </div>
    <div class="card-body">
      <form th:action="@{/calculate}" th:object="${studentResult}" method="post">
        <div class="mb-3">
          <label class="form-label">Student Name</label>
          <input type="text" th:field="*{studentName}" class="form-control" placeholder="Enter
student name" required>
        </div>
        <!-- Subject 1 -->
        <div class="row">
          <div class="col-md-6 mb-3">
            <label>Subject 1 - Insem (0-30)</label>
            <input type="number" th:field="*{sub1Insem}" class="form-control" min="0"
max="30" required>
          </div>
          <div class="col-md-6 mb-3">
            <label>Subject 1 - Endsem (0-70)</label>
            <input type="number" th:field="*{sub1Endsem}" class="form-control" min="0"
max="70" required>
```

```
        </div>
      </div>
      <!-- Subject 2 -->
      <div class="row">
        <div class="col-md-6 mb-3">
          <label>Subject 2 - Insem (0-30)</label>
          <input type="number" th:field="*{sub2Insem}" class="form-control" min="0"
max="30" required>
        </div>
        <div class="col-md-6 mb-3">
          <label>Subject 2 - Endsem (0-70)</label>
          <input type="number" th:field="*{sub2Endsem}" class="form-control" min="0"
max="70" required>
        </div>
      </div>
      <!-- Subject 3 -->
      <div class="row">
        <div class="col-md-6 mb-3">
          <label>Subject 3 - Insem (0-30)</label>
          <input type="number" th:field="*{sub3Insem}" class="form-control" min="0"
max="30" required>
        </div>
        <div class="col-md-6 mb-3">
          <label>Subject 3 - Endsem (0-70)</label>
          <input type="number" th:field="*{sub3Endsem}" class="form-control" min="0"
max="70" required>
        </div>
      </div>
      <!-- Subject 4 -->
      <div class="row">
        <div class="col-md-6 mb-3">
          <label>Subject 4 - Insem (0-30)</label>
          <input type="number" th:field="*{sub4Insem}" class="form-control" min="0"
max="30" required>
        </div>
        <div class="col-md-6 mb-3">
          <label>Subject 4 - Endsem (0-70)</label>
```

```
          <input type="number" th:field="*{sub4Endsem}" class="form-control" min="0"
max="70" required>
        </div>
       </div>
       <!-- Subject 5 -->
       <div class="row">
        <div class="col-md-6 mb-3">
         <label>Subject 5 - Insem (0-30)</label>
         <input type="number" th:field="*{sub5Insem}" class="form-control" min="0"
max="30" required>
        </div>
        <div class="col-md-6 mb-3">
         <label>Subject 5 - Endsem (0-70)</label>
         <input type="number" th:field="*{sub5Endsem}" class="form-control" min="0"
max="70" required>
        </div>
       </div>
       <div class="mt-4">
        <button type="submit" class="btn btn-primary">Store & Generate Certificate</button>
        <button type="reset" class="btn btn-secondary">Reset</button>
       </div>
     </form>
    </div>
  </div>
</div>
</body>
</html>
```

certificate.html

```
<!-- src/main/resources/templates/certificate.html -->
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8"/>
  <title>Certificate</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet"/>
  <style>
    .certificate {
```

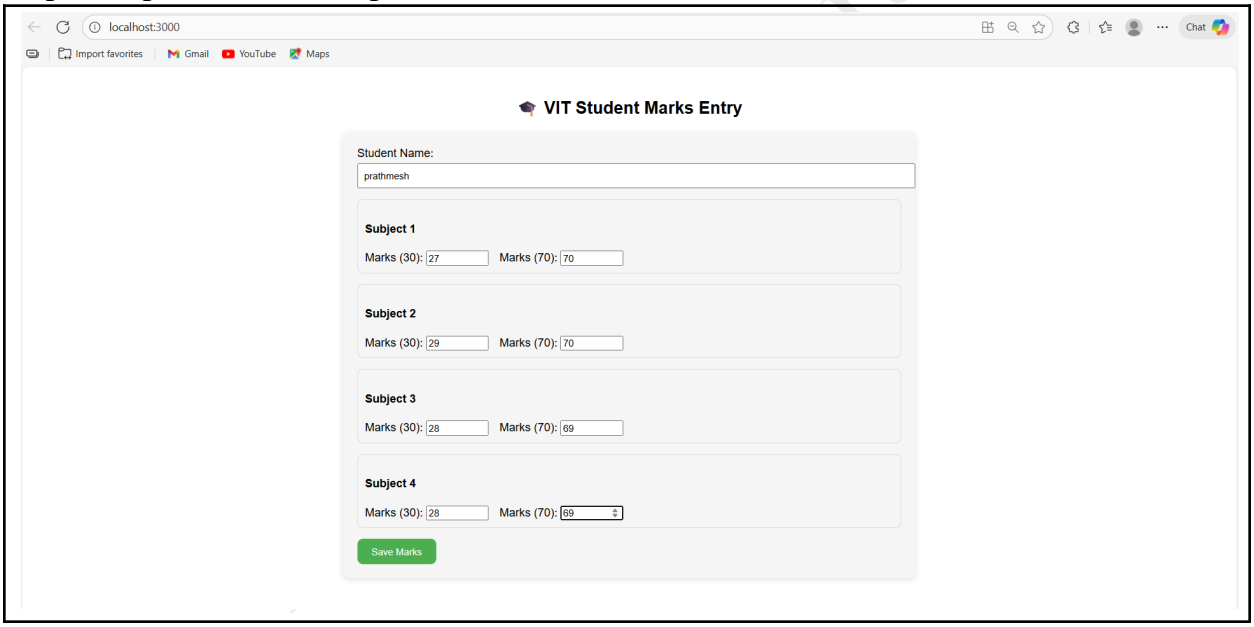
```
      border: 6px double #333;
      padding: 30px;
      background: #fff;
      margin: 30px auto;
      width: 800px;
    }
    @media print {
      .no-print { display: none; }
    }
  </style>
</head>
<body class="bg-light">
<div class="container">
  <div class="certificate">
    <div class="text-center mb-4">
      <h2>Institute Name</h2>
      <p>Certificate of Performance</p>
    </div>
    <div>
      <p><strong>Student Name:</strong> <span th:text="${result.studentName}">Student
Name</span></p>

      <table class="table table-bordered">
        <thead>
          <tr>
            <th>Subject</th>
            <th>Insem (30)</th>
            <th>Endsem (70)</th>
            <th>Total (100)</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>Subject 1</td>
            <td th:text="${result.sub1Insem}">0</td>
            <td th:text="${result.sub1Endsem}">0</td>
            <td th:text="${result.sub1Insem + result.sub1Endsem}">0</td>
          </tr>
```

```
        <tr>
          <td>Subject 2</td>
          <td th:text="${result.sub2Insem}">0</td>
          <td th:text="${result.sub2Endsem}">0</td>
          <td th:text="${result.sub2Insem + result.sub2Endsem}">0</td>
        </tr>
        <tr>
          <td>Subject 3</td>
          <td th:text="${result.sub3Insem}">0</td>
          <td th:text="${result.sub3Endsem}">0</td>
          <td th:text="${result.sub3Insem + result.sub3Endsem}">0</td>
        </tr>
        <tr>
          <td>Subject 4</td>
          <td th:text="${result.sub4Insem}">0</td>
          <td th:text="${result.sub4Endsem}">0</td>
          <td th:text="${result.sub4Insem + result.sub4Endsem}">0</td>
        </tr>
        <tr>
          <td>Subject 5</td>
          <td th:text="${result.sub5Insem}">0</td>
          <td th:text="${result.sub5Endsem}">0</td>
          <td th:text="${result.sub5Insem + result.sub5Endsem}">0</td>
        </tr>
      </tbody>
    </table>
    <p><strong>Total Marks (out of 500):</strong> <span
th:text="${result.totalMarks}">0</span></p>
    <p><strong>Percentage:</strong> <span th:text="${result.percentage}">0</span> %</p>
    <p><strong>CGPA:</strong> <span th:text="${result.cgpa}">0</span></p>
  </div>
  <div class="mt-4 d-flex justify-content-between">
    <div>
      <p>_____</p>
      <p>Exam Coordinator</p>
    </div>
    <div>
      <p>_____</p>
```

```
      <p>Principal</p>
    </div>
   </div>
   <div class="text-center mt-3 no-print">
    <button class="btn btn-success" onclick="window.print()">Print Certificate</button>
    <a class="btn btn-primary" th:href="@{/}">Enter New Result</a>
   </div>
  </div>
</div>
</body>
</html>
```

Step 6: Run DemoApplication as Spring Boot App.

Step 7 : Open browser: http://localhost:8080/



🏅 Certificate of Achievement

This is to certify that

prathmesh

has successfully completed the following subjects:

Subject	Marks (30)	Marks (70)	Total
Subject 1	27	70	97
Subject 2	29	70	99
Subject 3	28	69	97
Subject 4	28	69	97

🎯 Final CGPA: 9.75

Issued by VIT Examination Cell

Query 1

Limit to 1000 rows

```
    subject1_30 INT, subject1_70 INT, subject1_total INT,
    subject2_30 INT, subject2_70 INT, subject2_total INT,
    subject3_30 INT, subject3_70 INT, subject3_total INT,
    subject4_30 INT, subject4_70 INT, subject4_total INT,
    cgpa DECIMAL(5,2)
);
select * from student_marks
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	student_name	subject1_30	subject1_70	subject1_total	subject2_30	subject2_70	subject2_total	sub
1	prathmesh	27	67	94	27	67	94	27
2	prathmesh	27	67	94	27	67	94	27
3	prathmesh	27	67	94	27	67	94	27
4	prathmesh	27	67	94	27	67	94	27

Result Grid