

|  |  |
| --- | --- |
| Student Management System | |
|  |  |
| A Brief documentation of Student DATABASE Management System using Spring Boot |  |

**INDEX**

* Introduction
* Prerequisites
* Software Requirements
* Design and Architecture
* Implementation
* References

**INTRODUCTION**

Student Database Management System involves basic Create, Read, Update, Delete (CRUD) operations on the database. The application involves operations on database using Spring Boot.

The objective of the project is to design and develop a full stack application involving CRUD operations. The student database management system is the general system to start off with. The detail of a student is maintained in a relational database and a user interface allows the users to alter and update the contents of the database.

The requirements and prerequisites are mentioned further in this document. There are many other systems like product information management system, Employee management and many more that can be developed in the similar manner as this project.

The project provides independent user interfaces to perform each operation on the database. The actions performed in the user interface reflects same in the database as well. Naming conventions are user-friendly, and the source code is available in the mentioned link present in the reference section of the document.

Many parts of the backend code are a standard that is generically used. The user interface is subject to modification according to the user requirement. Various technologies are used in the project that are listed further.

**PREREQUISITES**

The expected audience must have a basic knowledge on the following technologies to obtain a clear understanding of the project.

* Java
* Spring Boot
* RDBMS
* JSP
* SQL
* HTML and CSS

**REQUIREMENTS**

Software requirements include:

* **JDK:**

The Java Development Kit is an essential software required for this project. This is helpful for compiling and executing the java code of the project.

The software is available in the official website of Oracle

<https://www.oracle.com/java/technologies/downloads/>

* **Spring Tool Suite:**

The spring tool suite is the platform that is used to develop spring

boot applications. The latest version or the version 3.19.12 is recommended.

The software can be downloaded from the following link:

<https://spring.io/tools>

* **MySQL Shell:**

The RDBMS used here is MySQL. The shell allows us to access the database. Even the workbench of MySQL is also capable of performing the required functions.

This can be downloaded and installed from:

<https://dev.mysql.com/downloads/installer/>

* **Browser:**

Any browser of user’s choice is suitable for the project. Latest version of browser is recommended.

**DESIGN AND ARCHITECTURE**

The project consists of basic model view controller architecture where all the database operation is done at the model and controller part and the view is to render the data to the user.

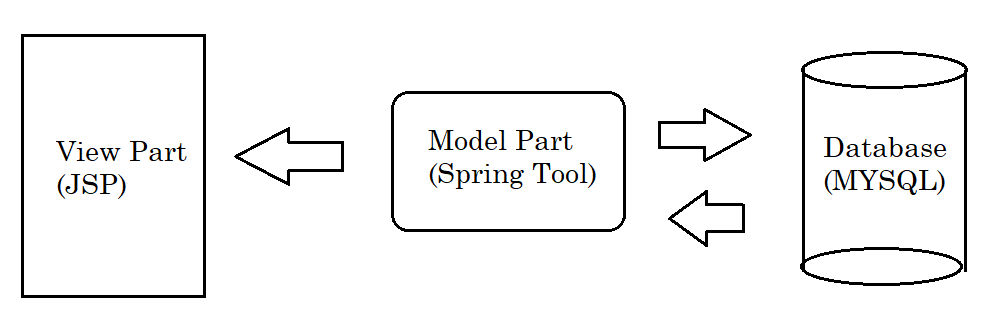


Fig 1.1: Schematic representation of the architecture

To create our application we have to create our database and the table corresponding to the application requirement in the Mysql Shell as follows

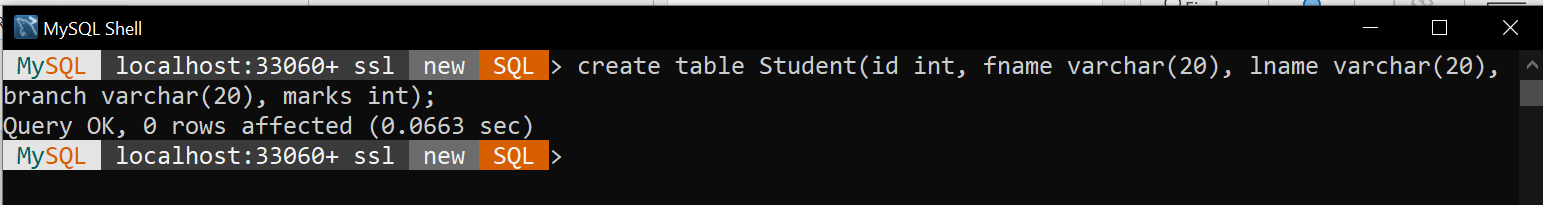


Fig 1.2: MySql table creation

The directory structure of the spring boot project is as shown in the below figure.

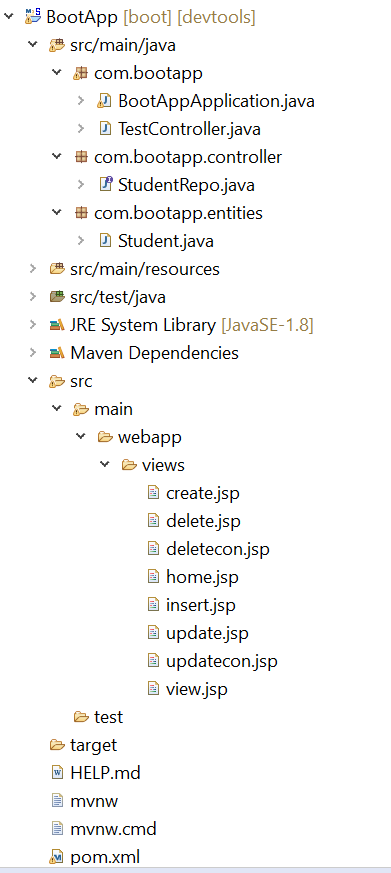


Fig 1.3: Directory Structure

The steps to create the boot project in STS is as follows:

1. In the panel click on New 🡪 Spring Stater Project.
2. Provide name of the project and other fields present there , then click Next.
3. In the search box type the required dependencies , select them and click on Next.
4. Click Next followed by Finish to create the project.
5. After the project is created , add subpackages and files as shown in the above figure.

The pom.xml which contains the dependencies of the project should contain the following

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<!-- https://mvnrepository.com/artifact/jstl/jstl -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<version>9.0.58</version>

</dependency>

The application.properties should contain the following code snippet.

spring.datasource.name = student

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

spring.datasource.username=root

spring.datasource.password=root

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

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

spring.jpa.hibernate.ddl-auto= update

spring.mvc.view.prefix=/views/

spring.mvc.view.suffix=.jsp

server.port = 8080

This file will be present in src/main/resources folder.

Graphical user interface, application, Word

Description automatically generatedThe main class that drives the application contains the default code and along with that the code to create a bean should be added as follows.

Fig 1.4: Main Boot Application

Similar to every spring boot project , the service class fetches the data from database and provides it to the model. In this project this is named as TestController.java.

The container interface that provides all the functionality on the database by extending the CrudRepository :

Graphical user interface, text, application

Description automatically generated

Fig 1.5: Extending CrudRepository

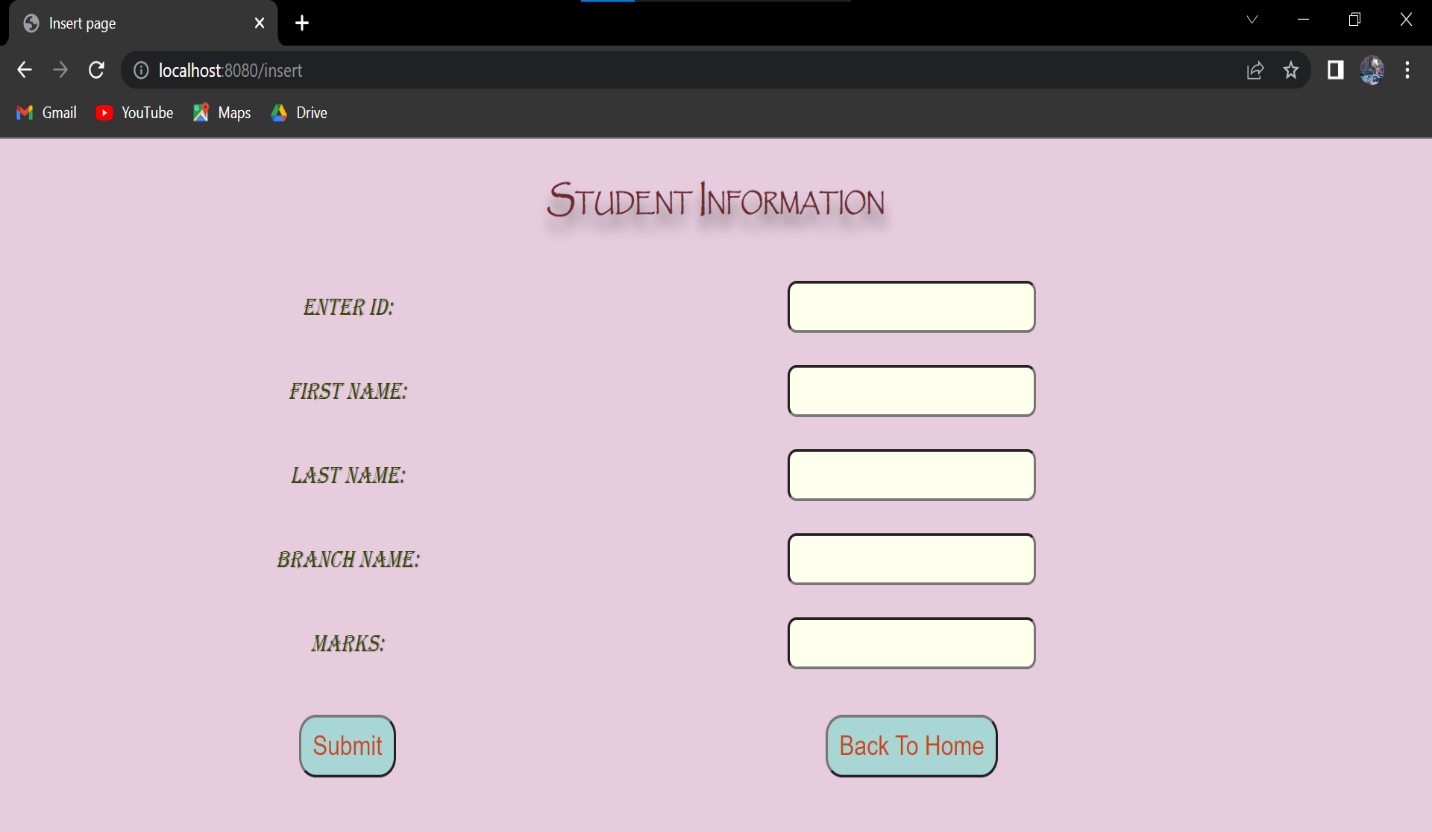
**IMPLEMENTATION**

The User interface of the home screen:

**Graphical user interface, text

Description automatically generated**

* *CREATE*



To add a new record , from the home screen we have to click on “Add Record” button to navigate to another screen containing html elements through which data can be received from user.

The controller code that handles the routing of the insert page :

@RequestMapping("/insert")

**public** String insert() {

**return** "insert";

}

@RequestMapping("/create")

**public** String create(Student s) {

repo.save(s);

System.***out***.println("Data added successfully");

**return** "create";

}

As soon as we enter the details and click on submit , the data will be added to database using save method of the CrudRepository on the create page.

The create page confirms the addition of new record

Graphical user interface, text

Description automatically generated

* *READ*

The retrieval of data is done using the JPQL query.

The retrieved data is sent to the view page by the model and the view renders it in a tabular format.

The query used to fetch data using JPQL can be seen in the Fig 1.5 present above.

A picture containing graphical user interface

Description automatically generated

The controller code that performs retrieval:

@RequestMapping("/view")

**public** String views(ModelMap mod) {

List<String> lst = repo.selectAll();

mod.addAttribute("list",lst);

**return** "view";

}

The JSTL is used to loop on the data sent from the controller and the table tag is used to display it.

* *UPDATE*

The user interface of the update part is almost similar to the insertion part , but guides the user to enter the data of a record that has to be updated.

The controller code that enables updation:

@RequestMapping("/update")

**public** String update() {

**return** "update";

}

@RequestMapping("/updatecon")

**public** String updatecon(Student s) {

repo.save(s);

System.***out***.println("Data Updated successfully");

**return** "updatecon";

}

Graphical user interface

Description automatically generated

As soon as the data gets updated another screen displays the confirmation.Graphical user interface, text, application

Description automatically generated

* *DELETE*

The deletion part involves user entering the “*id”* of the record and that will be received by the controller and it deleted the corresponding record in the database.

Graphical user interface, text, application, website

Description automatically generated

As the data gets deleted another screen confirms the action.

Graphical user interface

Description automatically generated

The controller code that performs deletion is:

@RequestMapping("/delete")

**public** String delete() {

**return** "delete";

}

@RequestMapping("/deletecon")

**public** String deletecon(**int** id) {

repo.deleteById(id);

System.***out***.println("Data Deleted");

**return** "deletecon";

}

**REFERENCES**

1. JSTL Documentation

<https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/c/forEach.html>

1. Source Code of the project

<https://github.com/sathwikm17/Crud-Spring>