Angular + Spring + MySql Full Stack

We are going to develop a CRUD (create-read-update-delete) web application. This application contains the student form that includes the CRUD features like add, view, delete, and update student. In this integration, we are using Spring Boot to handle the backend part and Angular to handle the frontend part.

Working of Application

- Once we deployed our application on the server, a student form generates at the web browser.
- The form facilitates to add and view students.
- On clicking add student link, the page redirects to create student form where we can add a student by filling the required details and submit them.
- Using view student link, we can fetch the details of the existing student. Here, each student also contains update and delete link.
- So, we can update the details of the student and also delete them from the database.
- Once completed, provide the URL http://localhost:4200/ at the web browser.

Tools to be used

- Use any IDE to develop the Spring and Hibernate project. It may be STS/Eclipse/Netbeans. Here, we are using STS (Spring Tool Suite).
- MySQL for the database.
- Use any IDE to develop the Angular project. It may be Visual Studio Code/Sublime. Here, we are using Visual Studio Code.
- Server: Apache Tomcat/JBoss/Glassfish/Weblogic/Websphere.

Technologies we used

Here, we are using the following technologies:

- SpringBoot 2
- Hibernate 5
- Angular 6
- MYSQL

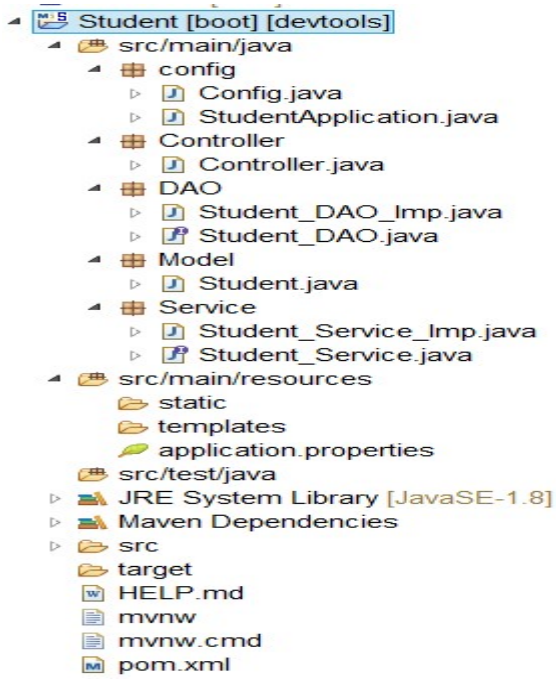
Create Database

Let's create a database **indigo**. There is no need to create a table as Hibernate automatically created it.

Angular + Spring + MySql Full Stack

Spring Module

Let's see the directory structure of Spring Boot we need to follow:



o develop a CRUD application, follow the below steps: -

Add dependencies to pom.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.main</groupId>
  <artifactId>Student</artifactId>
```

Angular + Spring + MySql Full Stack

```
<version>0.0.1-SNAPSHOT</version>
<name>Student</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
</properties>

<dependencies>

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

  <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>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

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

```
    </plugins>
  </build>

</project>
```

- Create the configuration class
  Instead of XML, we perform annotation-based configuration. So, we create a class Config.java and specify the required configuration in it. However, there is one more configuration class StudentApplication.java. This class is provided by Spring Boot automatically.q

Config.java

```
1.  package config;
2.
3.  import java.util.Properties;
4.  import javax.sql.DataSource;
5.  import org.springframework.beans.factory.annotation.Value;
6.  import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
7.  import org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguratio
    n;
8.  import org.springframework.context.annotation.Bean;
9.  import org.springframework.context.annotation.ComponentScan;
10. import org.springframework.context.annotation.ComponentScans;
11. import org.springframework.context.annotation.Configuration;
12. import org.springframework.jdbc.datasource.DriverManagerDataSource;
13. import org.springframework.orm.hibernate5.HibernateTransactionManager;
14. import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
15. import org.springframework.transaction.annotation.EnableTransactionManagement;
16. import org.springframework.web.servlet.view.InternalResourceViewResolver;
17.
18.
19. @Configuration
20. @EnableTransactionManagement
21. @EnableAutoConfiguration(exclude = { HibernateJpaAutoConfiguration.class})
22. @ComponentScans(value = { @ComponentScan("boot.entry"),
23.         @ComponentScan("Model"),
24.         @ComponentScan("Controller"),
25.         @ComponentScan("DAO"),
26.         @ComponentScan("Miscallaneous"),
27.         @ComponentScan("Service")})
28. public class Config {
```

```
29.
30.         @Value("${db.driver}")
31.         private String DB_DRIVER;
32.
33.         @Value("${db.password}")
34.         private String DB_PASSWORD;
35.
36.         @Value("${db.url}")
37.         private String DB_URL;
38.
39.         @Value("${db.username}")
40.         private String DB_USERNAME;
41.
42.         @Value("${hibernate.dialect}")
43.         private String HIBERNATE_DIALECT;
44.
45.         @Value("${hibernate.show_sql}")
46.         private String HIBERNATE_SHOW_SQL;
47.
48.         @Value("${hibernate.hbm2ddl.auto}")
49.         private String HIBERNATE_HBM2DDL_AUTO;
50.
51.         @Value("${entitymanager.packagesToScan}")
52.         private String ENTITYMANAGER_PACKAGES_TO_SCAN;
53.
54.         @Bean
55.         public LocalSessionFactoryBean sessionFactory() {
56.           LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
57.           sessionFactory.setDataSource(dataSource());
58.           sessionFactory.setPackagesToScan(ENTITYMANAGER_PACKAGES_TO_SCAN);
59.           Properties hibernateProperties = new Properties();
60.           hibernateProperties.put("hibernate.dialect", HIBERNATE_DIALECT);
61.           hibernateProperties.put("hibernate.show_sql", HIBERNATE_SHOW_SQL);
62.           hibernateProperties.put("hibernate.hbm2ddl.auto", HIBERNATE_HBM2DDL_AUT
    O);
63.           sessionFactory.setHibernateProperties(hibernateProperties);
64.           return sessionFactory;
65.         }
66.
67.         @Bean
68.         public DataSource dataSource() {
69.           DriverManagerDataSource dataSource = new DriverManagerDataSource();
70.           dataSource.setDriverClassName(DB_DRIVER);
```

```
71.                 dataSource.setUrl(DB_URL);
72.                 dataSource.setUsername(DB_USERNAME);
73.                 dataSource.setPassword(DB_PASSWORD);
74.                 return dataSource;
75.             }
76.
77.             @Bean
78.             public HibernateTransactionManager transactionManager() {
79.                 HibernateTransactionManager txManager = new HibernateTransactionManager()
    ;
80.                 txManager.setSessionFactory(sessionFactory().getObject());
81.                 return txManager;
82.             }
83.
84.             @Bean
85.             public InternalResourceViewResolver jspViewResolver() {
86.                 InternalResourceViewResolver resolver= new InternalResourceViewResolver();
87.                 resolver.setPrefix("/views/");
88.                 resolver.setSuffix(".jsp");
89.                 return resolver;
90.             }
91.
92.
93.
94.         }
```

StudentApplication.java

```
1.          package config;
2.
3.          import org.springframework.boot.SpringApplication;
4.          import org.springframework.boot.autoconfigure.SpringBootApplication;
5.
6.          @SpringBootApplication
7.          public class StudentApplication {
8.
9.              public static void main(String[] args) {
10.                 SpringApplication.run(StudentApplication.class, args);
11.             }
12.
13.         }
```

Angular + Spring + MySql Full Stack

- Create the entity class
  Here, we are creating an Entity/POJO (Plain Old Java Object) class.

Student.java

```
1.  package Model;
2.
3.  import javax.persistence.Entity;
4.  import javax.persistence.GeneratedValue;
5.  import javax.persistence.GenerationType;
6.  import javax.persistence.Id;
7.  import javax.persistence.Table;
8.
9.  @Entity
10. @Table(name="student")
11. public class Student {
12.     @Id
13.     @GeneratedValue(strategy=GenerationType.IDENTITY)
14.     private int student_id;
15.     private String student_name;
16.     private String student_email;
17.     private String student_branch;
18.     public int getStudent_id() {
19.         return student_id;
20.     }
21.     public void setStudent_id(int student_id) {
22.         this.student_id = student_id;
23.     }
24.     public String getStudent_name() {
25.         return student_name;
26.     }
27.     public void setStudent_name(String student_name) {
28.         this.student_name = student_name;
29.     }
30.     public String getStudent_email() {
31.         return student_email;
32.     }
33.     public void setStudent_email(String student_email) {
34.         this.student_email = student_email;
35.     }
36.     public String getStudent_branch() {
37.         return student_branch;
38.     }
```

```
39.         public void setStudent_branch(String student_branch) {
40.            this.student_branch = student_branch;
41.         }
42.
43.
44.      }
```

- Create the DAO interface
  Here, we are creating the DAO interface to perform database related operations.

Student_DAO.java

```
1.       package DAO;
2.
3.       import java.util.List;
4.
5.       import Model.Student;
6.
7.       public interface Student_DAO {
8.
9.          public boolean saveStudent(Student student);
10.         public List<Student> getStudents();
11.         public boolean deleteStudent(Student student);
12.         public List<Student> getStudentByID(Student student);
13.         public boolean updateStudent(Student student);
14.      }
```

- Create the DAO interface implementation class

Student_DAO_Imp.java

```
1.       package DAO;
2.
3.       import java.util.List;
4.
5.
6.       import org.hibernate.Session;
7.       import org.hibernate.SessionFactory;
8.       import org.hibernate.query.Query;
9.       import org.springframework.beans.factory.annotation.Autowired;
10.      import org.springframework.stereotype.Repository;
11.
12.
13.      import Model.Student;
```

```
14.
15.         @Repository
16.         public class Student_DAO_Imp  implements Student_DAO{
17.
18.             @Autowired
19.             private SessionFactory sessionFactory;
20.
21.             @Override
22.             public boolean saveStudent(Student student) {
23.                 boolean status=false;
24.                 try {
25.                     sessionFactory.getCurrentSession().save(student);
26.                     status=true;
27.                 } catch (Exception e) {
28.                     e.printStackTrace();
29.                 }
30.                 return status;
31.             }
32.
33.             @Override
34.             public List<Student> getStudents() {
35.                 Session currentSession = sessionFactory.getCurrentSession();
36.                 Query<Student> query=currentSession.createQuery("from Student", Student.class);

37.                 List<Student> list=query.getResultList();
38.                 return list;
39.             }
40.
41.             @Override
42.             public boolean deleteStudent(Student student) {
43.                 boolean status=false;
44.                 try {
45.                     sessionFactory.getCurrentSession().delete(student);
46.                     status=true;
47.                 } catch (Exception e) {
48.                     e.printStackTrace();
49.                 }
50.                 return status;
51.             }
52.
53.             @Override
54.             public List<Student> getStudentByID(Student student) {
55.                 Session currentSession = sessionFactory.getCurrentSession();
```

```
56.             Query<Student> query=currentSession.createQuery("from Student where student_id=:student_id", Student.class);
57.             query.setParameter("student_id", student.getStudent_id());
58.             List<Student> list=query.getResultList();
59.             return list;
60.         }
61.
62.         @Override
63.         public boolean updateStudent(Student student) {
64.             boolean status=false;
65.             try {
66.                 sessionFactory.getCurrentSession().update(student);
67.                 status=true;
68.             } catch (Exception e) {
69.                 e.printStackTrace();
70.             }
71.             return status;
72.         }
73.
74.
75.
76.     }
```

- Create the service layer interface

Here, we are creating a service layer interface that acts as a bridge between DAO and Entity classes.

Student_Service.java

```
1.      package Service;
2.
3.      import java.util.List;
4.      import Model.Student;
5.
6.      public interface Student_Service {
7.
8.
9.          public boolean saveStudent(Student student);
10.         public List<Student> getStudents();
11.         public boolean deleteStudent(Student student);
12.         public List<Student> getStudentByID(Student student);
13.         public boolean updateStudent(Student student);
```

```
}
```

- Create the service layer implementation class

Student_Service_Imp.java

```
package Service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import DAO.Student_DAO;
import Model.Student;

@Service
@Transactional
public class Student_Service_Imp implements Student_Service {

  @Autowired
  private Student_DAO studentdao;

  @Override
  public boolean saveStudent(Student student) {
    return studentdao.saveStudent(student);
  }

  @Override
  public List<Student> getStudents() {
    return studentdao.getStudents();
  }

  @Override
  public boolean deleteStudent(Student student) {
    return studentdao.deleteStudent(student);
  }

  @Override
  public List<Student> getStudentByID(Student student) {
    return studentdao.getStudentByID(student);
  }

  @Override
```

```
38.         public boolean updateStudent(Student student) {
39.           return studentdao.updateStudent(student);
40.         }
41.
42.       }
```

- Create the controller class

Controller.java

```
1.        package Controller;
2.
3.        import java.util.List;
4.        import org.springframework.beans.factory.annotation.Autowired;
5.        import org.springframework.web.bind.annotation.CrossOrigin;
6.        import org.springframework.web.bind.annotation.DeleteMapping;
7.        import org.springframework.web.bind.annotation.GetMapping;
8.        import org.springframework.web.bind.annotation.PathVariable;
9.        import org.springframework.web.bind.annotation.PostMapping;
10.       import org.springframework.web.bind.annotation.RequestBody;
11.       import org.springframework.web.bind.annotation.RequestMapping;
12.       import org.springframework.web.bind.annotation.RestController;
13.
14.       import Model.Student;
15.       import Service.Student_Service;
16.
17.       @RestController
18.       @CrossOrigin(origins="http://localhost:4200")
19.       @RequestMapping(value="/api")
20.       public class Controller {
21.
22.         @Autowired
23.         private Student_Service studentservice;
24.
25.         @PostMapping("save-student")
26.         public boolean saveStudent(@RequestBody Student student) {
27.           return studentservice.saveStudent(student);
28.
29.         }
30.
31.         @GetMapping("students-list")
32.         public List<Student> allstudents() {
33.           return studentservice.getStudents();
```

Angular + Spring + MySql Full Stack

```
34.
35.             }
36.
37.             @DeleteMapping("delete-student/{student_id}")
38.             public boolean deleteStudent(@PathVariable("student_id") int student_id,Student st
    udent) {
39.                 student.setStudent_id(student_id);
40.                 return studentservice.deleteStudent(student);
41.             }
42.
43.             @GetMapping("student/{student_id}")
44.             public List<Student> allstudentByID(@PathVariable("student_id") int student_id,Stud
    ent student) {
45.                  student.setStudent_id(student_id);
46.                  return studentservice.getStudentByID(student);
47.
48.             }
49.
50.             @PostMapping("update-student/{student_id}")
51.             public boolean updateStudent(@RequestBody Student student,@PathVariable("stud
    ent_id") int student_id) {
52.                 student.setStudent_id(student_id);
53.                 return studentservice.updateStudent(student);
54.             }
55.         }
```

- Edit application.properties file
  Here, we are editing the **application.properties** file present inside the **src/main/resources** folder. The following file contains the configuration properties.

application.properties

```
1.          # Database
2.          db.driver= com.mysql.jdbc.Driver
3.          db.url= jdbc:mysql://localhost:3306/indigo
4.          db.username=root
5.          db.password=
6.
7.          # Hibernate
8.          hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
9.          hibernate.show_sql=true
10.         hibernate.hbm2ddl.auto=update
11.         entitymanager.packagesToScan=Model
```

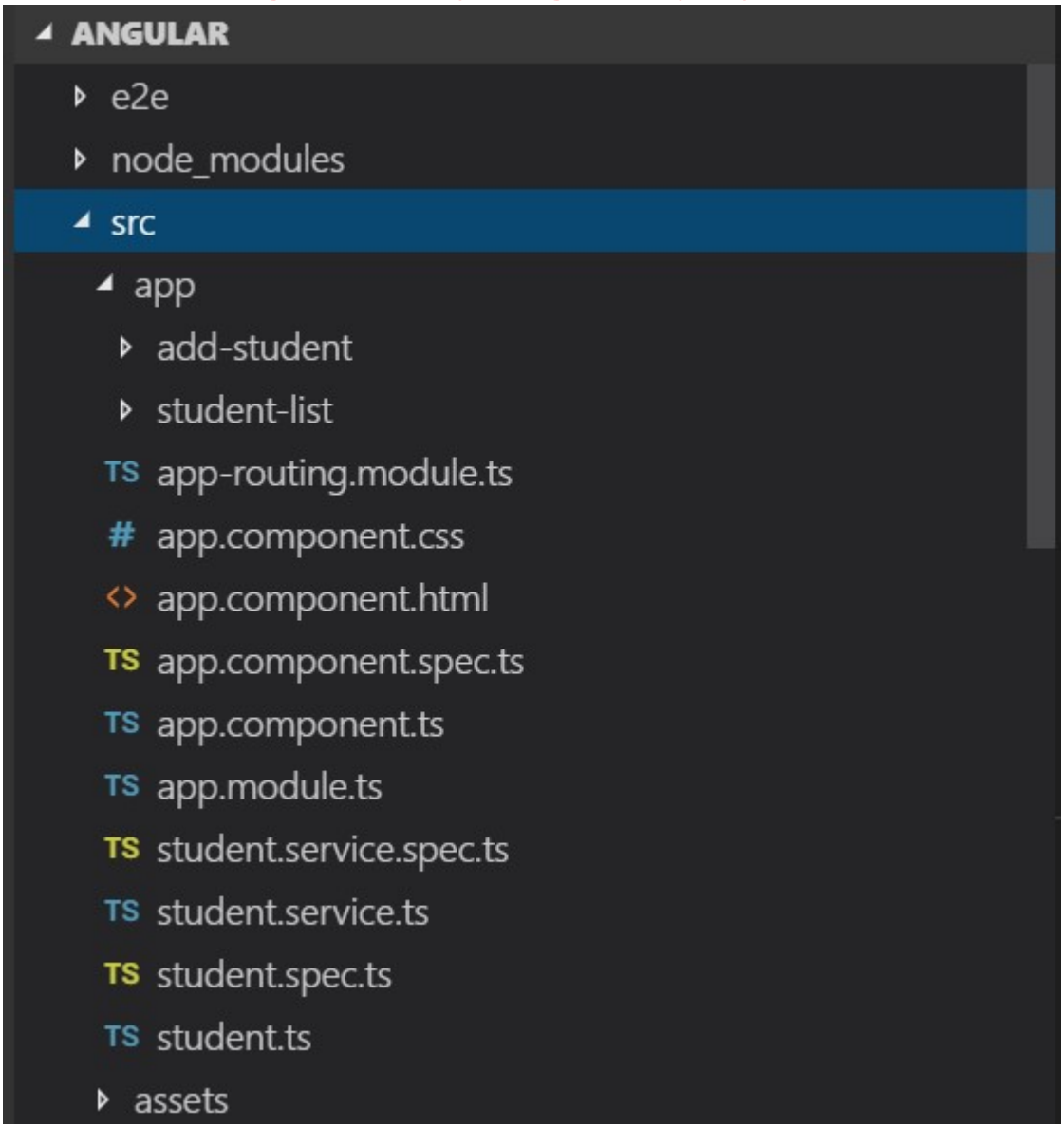
Angular + Spring + MySql Full Stack

Angular Coding Section Starting (Front End Application)

Angular Module

Let's see the directory structure of Angular we need to follow:

Angular + Spring + MySql Full Stack



- Create an Angular project

Let's create an Angular project by using the following command:

ng new Angular

Angular + Spring + MySql Full Stack

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\developer\Documents\Gaurav>ng new Angular
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS    [ http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax ]

? Which stylesheet format would you like to use? CSS
CREATE Angular/angular.json (3433 bytes)
CREATE Angular/package.json (1280 bytes)
CREATE Angular/README.md (1024 bytes)
CREATE Angular/tsconfig.json (438 bytes)
CREATE Angular/tslint.json (1985 bytes)
CREATE Angular/.editorconfig (246 bytes)
CREATE Angular/.gitignore (629 bytes)
CREATE Angular/browserslist (429 bytes)
CREATE Angular/karma.conf.js (1019 bytes)
CREATE Angular/tsconfig.app.json (210 bytes)
CREATE Angular/tsconfig.spec.json (270 bytes)
CREATE Angular/src/favicon.ico (5430 bytes)
CREATE Angular/src/index.html (294 bytes)
CREATE Angular/src/main.ts (372 bytes)
CREATE Angular/src/polyfills.ts (2838 bytes)
```

```
C:\Windows\System32\cmd.exe
> core-js@2.6.9 postinstall C:\Users\developer\Documents\Gaurav\Angular\node_mod
ules\karma\node_modules\core-js
> node scripts/postinstall || echo "ignore"


> @angular/cli@8.0.3 postinstall C:\Users\developer\Documents\Gaurav\Angular\nod
e_modules\@angular\cli
> node ./bin/postinstall/script.js

npm WARN rollback Rolling back needle@2.3.0 failed (this is probably harmless):
EPERM: operation not permitted, rmdir 'C:\Users\developer\Documents\Gaurav\Angul
ar\node_modules\fsevents\node_modules'
npm WARN rollback Rolling back readable-stream@2.3.6 failed (this is probably ha
rmless): EPERM: operation not permitted, lstat 'C:\Users\developer\Documents\Gau
rav\Angular\node_modules\fsevents'
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fse
vents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@
1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"}
)

added 1012 packages from 1041 contributors in 215.099s
'git' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\developer\Documents\Gaurav>
```

Here, **Angular** is the name of the project.

Install Bootstrap CSS framework

Use the following command to install bootstrap in the project.

npm install bootstrap@3.3.7 --save

Now, include the following code in the style.css file.

```
@import "~bootstrap/dist/css/bootstrap.css";
```

Angular + Spring + MySql Full Stack

Install Angular-DataTable

Use the following command to install angular datatable in the project.

npm install angular-datatable –save

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: cmd

C:\Users\developer\Documents\Gaurav\Angular>npm install angular-datatable --save
npm WARN angular-datatable@2.2.1 requires a peer of @angular/core@^2.4.0 but none is installed
. You must install peer dependencies yourself.
npm WARN angular-datatable@2.2.1 requires a peer of @angular/common@^2.4.0 but none is install
ed. You must install peer dependencies yourself.
npm WARN angular-datatable@2.2.1 requires a peer of @angular/forms@^2.4.0 but none is installe
d. You must install peer dependencies yourself.
npm WARN angular-datatable@2.2.1 requires a peer of rxjs@^5.0.1 but none is installed. You mus
t install peer dependencies yourself.
npm WARN angular-datatable@2.2.1 requires a peer of zone.js@^0.7.2 but none is installed. You
must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted
{"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ angular-datatable@2.2.1
added 1 package from 1 contributor and audited 19006 packages in 17.535s
found 0 vulnerabilities


C:\Users\developer\Documents\Gaurav\Angular>
```

It is required to include **DataTableModule** in imports array of **app.module.ts** file.

- Generate Components
  Open the project in visual studio and then use the following command to generate Angular components:
  ng g c AddStudent
  ng g c StudentList

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: cmd

C:\Users\developer\Documents\Gaurav\Angular>ng g c AddStudent
CREATE src/app/add-student/add-student.component.html (30 bytes)
CREATE src/app/add-student/add-student.component.spec.ts (657 bytes)
CREATE src/app/add-student/add-student.component.ts (288 bytes)
CREATE src/app/add-student/add-student.component.css (0 bytes)
UPDATE src/app/app.module.ts (493 bytes)

C:\Users\developer\Documents\Gaurav\Angular>ng g c StudentList
CREATE src/app/student-list/student-list.component.html (31 bytes)
CREATE src/app/student-list/student-list.component.spec.ts (664 bytes)
CREATE src/app/student-list/student-list.component.ts (292 bytes)
CREATE src/app/student-list/student-list.component.css (0 bytes)
UPDATE src/app/app.module.ts (597 bytes)

C:\Users\developer\Documents\Gaurav\Angular>
```

Angular + Spring + MySql Full Stack

Let's also create a service class by using the following command: -

ng g s Student

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: cmd

C:\Users\developer\Documents\Gaurav\Angular>ng g s Student
CREATE src/app/student.service.spec.ts (338 bytes)
CREATE src/app/student.service.ts (136 bytes)

C:\Users\developer\Documents\Gaurav\Angular>
```

Edit the **app.module.ts** file

- **Import Routing** - Here, we are importing routing file (app-routing.module.ts) and include it in imports array.
- **Import ReactiveFormsModule** - Here, we are importing **ReactiveFormsModule** for reactive forms and specify it in imports array.
- **Import HttpModule** - Here, we are importing **HttpModule** for server requests and specifying it in imports array.
- **Register Service class** - Here, we are mentioning the service class in provider's array.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import {DataTablesModule} from 'angular-datatables';
import { StudentListComponent } from './student-list/student-list.component';
import { AddStudentComponent } from './add-student/add-student.component';

@NgModule({
 declarations: [
  AppComponent,
  StudentListComponent,
  AddStudentComponent,
 ],
 imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  ReactiveFormsModule,
  HttpClientModule,
  DataTablesModule
```

Angular + Spring + MySql Full Stack

```
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Edit the **app-routing.module.ts** file

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { StudentListComponent } from './student-list/student-list.component';
import { AddStudentComponent } from './add-student/add-student.component';

const routes: Routes = [
  { path: '', redirectTo: 'view-student', pathMatch: 'full' },
  { path: 'view-student', component: StudentListComponent },
  { path: 'add-student', component: AddStudentComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Edit the **app.component.html** file

```
<div  class="container-fluid">
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
   <ul class="navbar-nav">
    <li class="nav-item ">
      <a routerLink="view-student" class="nav-link" class="btn btn-primary active" role="button" >View Student</a>
    </li>
    <li class="nav-item">
      <a  routerLink="add-student" class="nav-link" class="btn btn-primary active" role="button" >Add Student</a>
    </li>
   </ul>
  </nav>
   <router-outlet></router-outlet>
</div>
```

Angular + Spring + MySql Full Stack

- Create the **Student.ts** class

Let's create a class by using the following command: -

ng g class Student

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: cmd

C:\Users\developer\Documents\Gaurav\Angular>ng g class Student
CREATE src/app/student.spec.ts (158 bytes)
CREATE src/app/student.ts (25 bytes)

C:\Users\developer\Documents\Gaurav\Angular>
```

Now, specify the required fields within the **Student** class.

```
export class Student {

   student_id:number;
   student_name:String;
   student_email:String;
   student_branch:String;
}
```

The purpose of this class is to map the specified fields with the fields of Spring entity class.

- Edit the **student.service.ts** file

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class StudentService {

  private baseUrl = 'http://localhost:8080/api/';

  constructor(private http:HttpClient) { }

  getStudentList(): Observable<any> {
    return this.http.get(`${this.baseUrl}`+'students-list');
```

```
17.         }
18.
19.         createStudent(student: object): Observable<object> {
20.           return this.http.post(`${this.baseUrl}`+'save-student', student);
21.         }
22.
23.         deleteStudent(id: number): Observable<any> {
24.           return this.http.delete(`${this.baseUrl}/delete-
student/${id}`, { responseType: 'text' });
25.         }
26.
27.         getStudent(id: number): Observable<Object> {
28.           return this.http.get(`${this.baseUrl}/student/${id}`);
29.         }
30.
31.         updateStudent(id: number, value: any): Observable<Object> {
32.           return this.http.post(`${this.baseUrl}/update-student/${id}`, value);
33.         }
34.
35.       }
```

- Edit the **add-student.component.ts** file

```
1.        import { Component, OnInit } from '@angular/core';
2.        import { StudentService } from '../student.service';
3.        import {FormControl,FormGroup,Validators} from '@angular/forms';
4.        import { Student } from '../student';
5.        @Component({
6.          selector: 'app-add-student',
7.          templateUrl: './add-student.component.html',
8.          styleUrls: ['./add-student.component.css']
9.        })
10.       export class AddStudentComponent implements OnInit {
11.
12.         constructor(private studentservice:StudentService) { }
13.
14.         student : Student=new Student();
15.         submitted = false;
16.
17.         ngOnInit() {
18.           this.submitted=false;
19.         }
20.
```

```
21.         studentsaveform=new FormGroup({
22.           student_name:new FormControl('' , [Validators.required , Validators.minLength(5) ] ),

23.           student_email:new FormControl('',[Validators.required,Validators.email]),
24.           student_branch:new FormControl()
25.         });
26.
27.         saveStudent(saveStudent){
28.           this.student=new Student();
29.           this.student.student_name=this.StudentName.value;
30.           this.student.student_email=this.StudentEmail.value;
31.           this.student.student_branch=this.StudentBranch.value;
32.           this.submitted = true;
33.           this.save();
34.         }
35.
36.
37.
38.         save() {
39.           this.studentservice.createStudent(this.student)
40.             .subscribe(data => console.log(data), error => console.log(error));
41.           this.student = new Student();
42.         }
43.
44.         get StudentName(){
45.           return this.studentsaveform.get('student_name');
46.         }
47.
48.         get StudentEmail(){
49.           return this.studentsaveform.get('student_email');
50.         }
51.
52.         get StudentBranch(){
53.           return this.studentsaveform.get('student_branch');
54.         }
55.
56.         addStudentForm(){
57.           this.submitted=false;
58.           this.studentsaveform.reset();
59.         }
60.       }
```

- Edit the **add-student.component.html** file

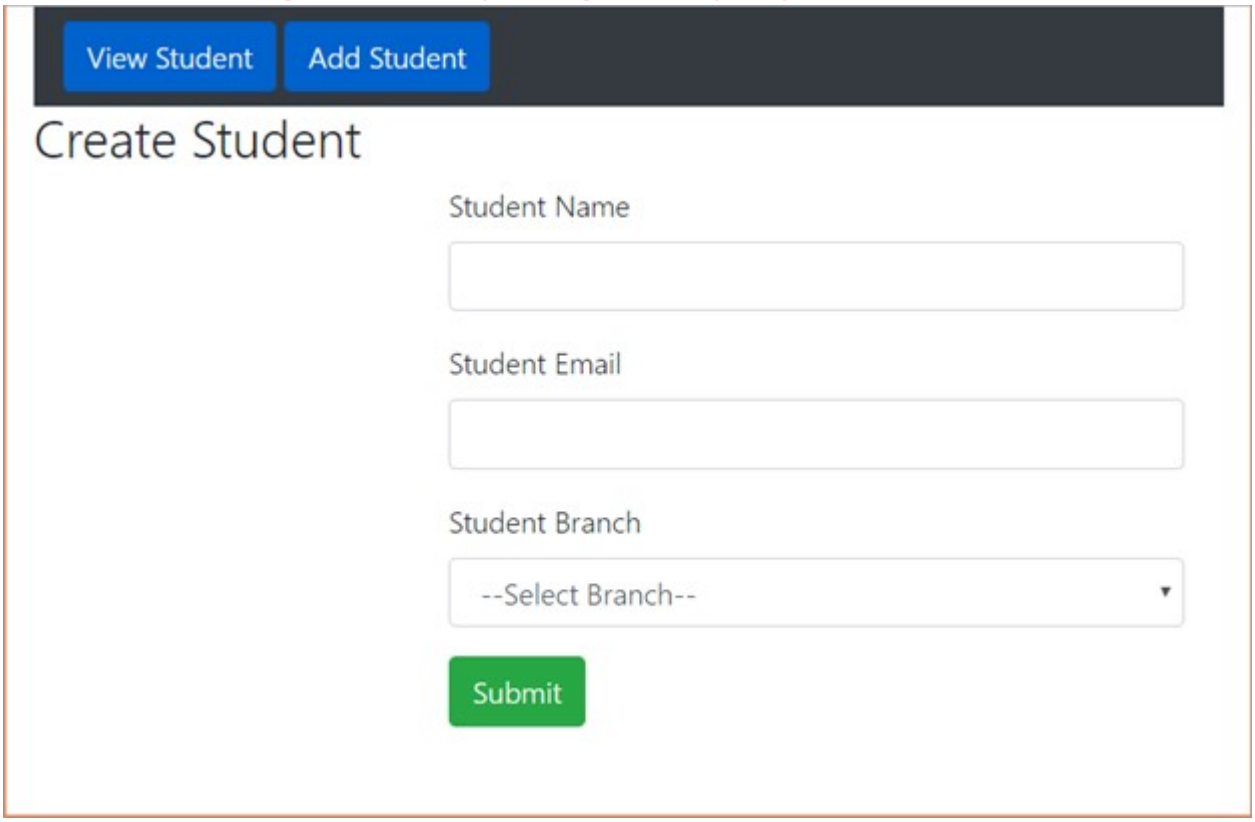
Angular + Spring + MySql Full Stack

```
1.          <h3>Create Student</h3>
2.          <div class="row">
3.            <div class="col-sm-4"></div>
4.            <div class="col-sm-4" >
5.              <div [hidden]="submitted" style="width: 400px;">
6.                <form [formGroup]="studentsaveform" #savestudent (ngSubmit)="saveStudent(save
    Student)">
7.                  <div class="form-group">
8.                    <label for="name">Student Name</label>
9.                    <input type="text" class="form-
    control"  formControlName="student_name" data-toggle="tooltip"
10.                     data-placement="right" title="Enter Student Name" >
11.                   <div class="alert alert-
    danger" *ngIf = "(StudentName.touched) && (StudentName.invalid)"
12.                     style="margin-top: 5px;">
13.                       <span *ngIf="StudentName.errors.required">Student Name is Required</spa
    n>
14.                       <span *ngIf = "StudentName.errors.minlength">
15.                         MinLength Error
16.                       </span>
17.                     </div>
18.                 </div>
19.
20.                 <div class="form-group">
21.                   <label for="name">Student Email</label>
22.                   <input type="text" class="form-control" formControlName="student_email"
23.                     data-toggle="tooltip" data-placement="right" title="Enter Student Email">
24.                     <div class="alert alert-
    danger" *ngIf = "(StudentEmail.touched) && (StudentEmail.invalid)"
25.                     style="margin-top: 5px;">
26.                       <span *ngIf="StudentEmail.errors.required">Student Email is Required</span
    >
27.                       <span *ngIf = "StudentEmail.errors.email">
28.                         Invalid Email Format
29.                       </span>
30.                     </div>
31.                 </div>
32.
33.                 <div class="form-group">
34.                   <label for="branch">Student Branch</label>
35.                     <select class="form-control" formControlName="student_branch" data-
    toggle="tooltip"
36.                         data-placement="right" title="Select Student Branch">
```

```
37.                     <option value="null">--Select Branch--</option>
38.                    <option value="B-Tech">B-Tech</option>
39.                    <option value="BCA">BCA</option>
40.                    <option value="MCA">MCA</option>
41.                    <option value="M-Tech">M-Tech</option>
42.                   </select>
43.                </div>
44.
45.                <button type="submit" class="btn btn-success">Submit</button>
46.              </form>
47.            </div>
48.            </div>
49.            <div class="col-sm-4"></div>
50.          </div>
51.          <div class="row">
52.            <div class="col-sm-4"></div>
53.            <div class="col-sm-4">
54.              <div [hidden]="!submitted">
55.                <h4>Student Added SuccessFully!</h4>
56.                <button (click)="addStudentForm()" class='btn btn-
primary'>Add More Student</button>
57.              </div>
58.            </div>
59.            <div class="col-sm-4"></div>
60.          </div>
```

On clicking **Add Student**, the following page generates:

Angular + Spring + MySql Full Stack



Now, fill the required details and submit them to add student.

- Edit the **student-list.component.ts** file

```
import { Component, OnInit } from '@angular/core';
import { StudentService } from '../student.service';
import { Student } from '../student';
import { Observable,Subject } from "rxjs";

import {FormControl,FormGroup,Validators} from '@angular/forms';

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.css']
})
export class StudentListComponent implements OnInit {

 constructor(private studentservice:StudentService) { }

  studentsArray: any[] = [];
```

```
18.         dtOptions: DataTables.Settings = {};
19.         dtTrigger: Subject<any>= new Subject();
20.
21.         students: Observable<Student[]>;
22.         student : Student=new Student();
23.         deleteMessage=false;
24.         studentlist:any;
25.         isupdated = false;
26.
27.
28.         ngOnInit() {
29.           this.isupdated=false;
30.           this.dtOptions = {
31.             pageLength: 6,
32.             stateSave:true,
33.             lengthMenu:[[6, 16, 20, -1], [6, 16, 20, "All"]],
34.             processing: true
35.           };
36.           this.studentservice.getStudentList().subscribe(data =>{
37.           this.students =data;
38.           this.dtTrigger.next();
39.           })
40.         }
41.
42.         deleteStudent(id: number) {
43.           this.studentservice.deleteStudent(id)
44.             .subscribe(
45.               data => {
46.                 console.log(data);
47.                 this.deleteMessage=true;
48.                 this.studentservice.getStudentList().subscribe(data =>{
49.                   this.students =data
50.                   })
51.               },
52.               error => console.log(error));
53.         }
54.
55.         updateStudent(id: number){
56.           this.studentservice.getStudent(id)
57.             .subscribe(
58.               data => {
59.                 this.studentlist=data
60.               },
```

```
61.             error => console.log(error));
62.         }
63.
64.         studentupdateform=new FormGroup({
65.           student_id:new FormControl(),
66.           student_name:new FormControl(),
67.           student_email:new FormControl(),
68.           student_branch:new FormControl()
69.         });
70.
71.         updateStu(updstu){
72.           this.student=new Student();
73.          this.student.student_id=this.StudentId.value;
74.          this.student.student_name=this.StudentName.value;
75.          this.student.student_email=this.StudentEmail.value;
76.          this.student.student_branch=this.StudentBranch.value;
77.          console.log(this.StudentBranch.value);
78.
79.
80.          this.studentservice.updateStudent(this.student.student_id,this.student).subscribe(
81.           data => {
82.             this.isupdated=true;
83.             this.studentservice.getStudentList().subscribe(data =>{
84.               this.students =data
85.               })
86.           },
87.           error => console.log(error));
88.         }
89.
90.         get StudentName(){
91.           return this.studentupdateform.get('student_name');
92.         }
93.
94.         get StudentEmail(){
95.           return this.studentupdateform.get('student_email');
96.         }
97.
98.         get StudentBranch(){
99.           return this.studentupdateform.get('student_branch');
100.        }
101.
102.        get StudentId(){
103.          return this.studentupdateform.get('student_id');
```

```
104.        }
105.
106.        changeisUpdate(){
107.          this.isupdated=false;
108.        }
109.       }
```

- Edit the **student-list.component.html** file

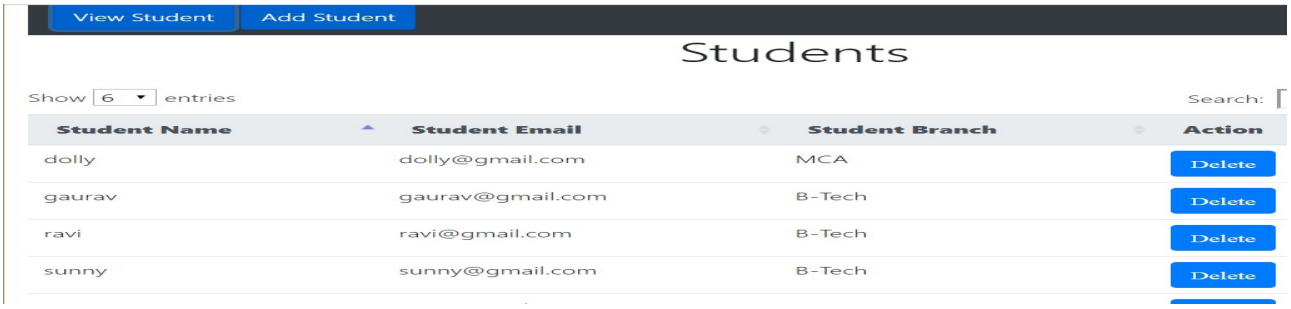
```
1.          <div class="panel panel-default">
2.           <div class="panel-heading">
3.              <h1 style="text-align: center">Students</h1><br>
4.              <div class="row" [hidden]="!deleteMessage">
5.
6.                      <div class="col-sm-4"></div>
7.                      <div class="col-sm-4">
8.                           <div class="alert alert-info alert-dismissible">
9.                                <button type="button" class="close" data-
   dismiss="alert">×</button>
10.                               <strong>Student Data Deleted</strong>
11.                          </div>
12.                     </div>
13.                     <div class="col-sm-4"></div>
14.                 </div>
15.             </div>
16.
17.
18.          <div class="panel-body">
19.             <table  class="table table-hover table-sm" datatable [dtOptions]="dtOptions"
20.             [dtTrigger]="dtTrigger"  >
21.                <thead class="thead-light">
22.                   <tr>
23.                      <th>Student Name</th>
24.                      <th>Student Email</th>
25.                      <th>Student Branch</th>
26.                      <th>Action</th>
27.
28.                   </tr>
29.                </thead>
30.                <tbody>
31.                    <tr *ngFor="let student of students ">
32.                      <td>{{student.student_name}}</td>
33.                      <td>{{student.student_email}}</td>
```

Angular + Spring + MySql Full Stack

```
34.                     <td>{{student.student_branch}}</td>
35.                     <td><button (click)="deleteStudent(student.student_id)" class='btn btn-
primary'><i class="fa fa-futboll-0">Delete</i></button>
36.                      <button (click)="updateStudent(student.student_id)" class='btn btn-info'
37.                      data-toggle="modal" data-target="#myModal">Update</button>
38.                     </td>
39.                    </tr>
40.                 </tbody><br>
41.              </table>
42.            </div>
43.          </div>
44.
45.          <div class="modal" id="myModal">
46.               <div class="modal-dialog">
47.                <div class="modal-content">
48.                    <form [formGroup]="studentupdateform" #updstu (ngSubmit)="updateStu(up
dstu)">
49.                  <!-- Modal Header -->
50.                  <div class="modal-header">
51.                   <h4 class="modal-title" style="text-align: center">Update Student</h4>
52.
53.                  </div>
54.
55.                  <!-- Modal body -->
56.                  <div class="modal-body" *ngFor="let student of studentlist " >
57.                    <div [hidden]="isupdated">
58.
59.                       <input type="hidden" class="form-
control"  formControlName="student_id" [(ngModel)]="student.student_id">
60.                            <div class="form-group">
61.                              <label for="name">Student Name</label>
62.                              <input type="text" class="form-
control"  formControlName="student_name" [(ngModel)]="student.student_name"  >
63.                            </div>
64.
65.                            <div class="form-group">
66.                              <label for="name">Student Email</label>
67.                              <input type="text" class="form-
control" formControlName="student_email" [(ngModel)]="student.student_email">
68.                            </div>
69.
70.                            <div class="form-group">
71.                              <label for="name">Student Branch</label>
```

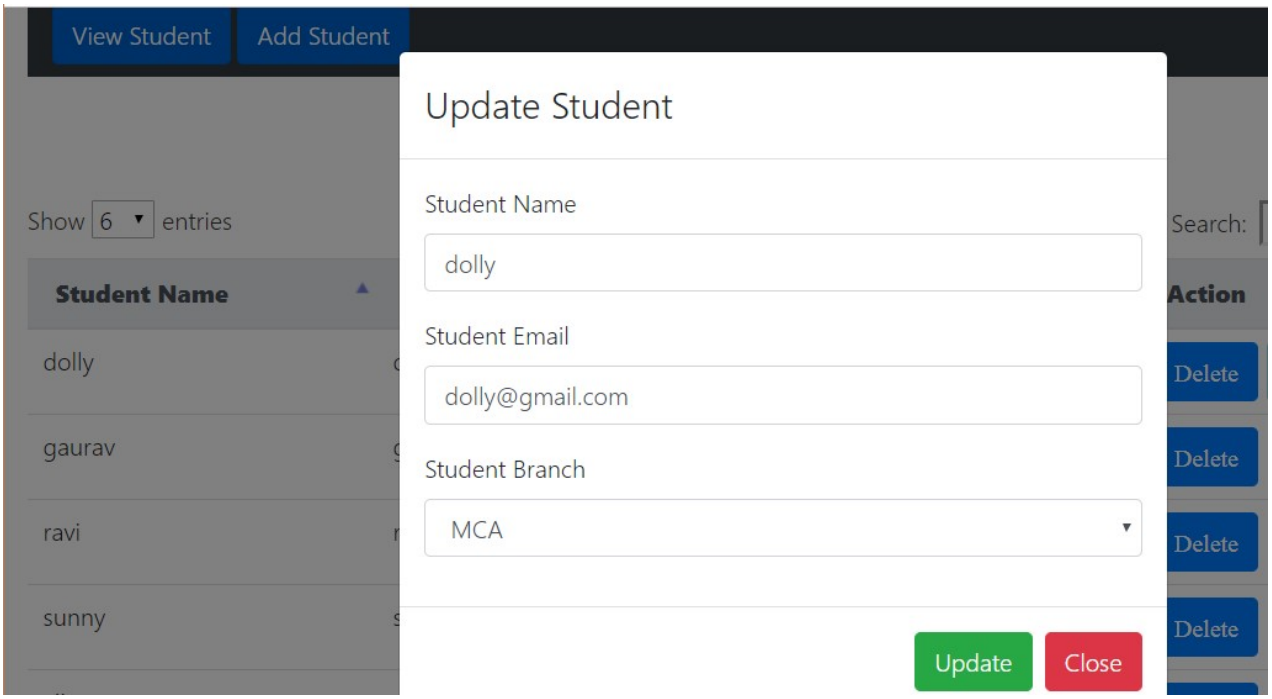
```
72.                                 <select class="form-
control" formControlName="student_branch" required>
73.                                  <option value="B-Tech" [selected]="'B-
Tech' == student.student_branch">B-Tech</option>
74.                                  <option value="BCA" [selected]="'BCA' == student.student_branch"
>BCA</option>
75.                                  <option value="MCA" [selected]="'MCA' == student.student_branch
" >MCA</option>
76.                                  <option value="M-Tech" [selected]="'M-
Tech' == student.student_branch">M-Tech</option>
77.                                 </select>
78.                             </div>
79.                         </div>
80.                         <div [hidden]="!isupdated">
81.                           <h4>Student Detail Updated!</h4>
82.                         </div>
83.
84.                     </div>
85.
86.                     <!-- Modal footer -->
87.                     <div class="modal-footer" >
88.                      <button type="submit" class="btn btn-
success" [hidden]="isupdated">Update</button>
89.                      <button type="button" class="btn btn-danger" data-
dismiss="modal" (click)="changeisUpdate()">Close</button>
90.                     </div>
91.
92.                   </form>
93.                    </div>
94.                   </div>
95.                 </div>
```

On clicking **View Student**, the following page generates:



Angular + Spring + MySql Full Stack

On clicking **Update Student**, the following bootstrap modal appears:



Here, we can update the details of the existing student.

On clicking **Delete**, the existing student is deleted from the database. Let's see the result after deleting the particular student.