

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**20 Marks Non CIE Component**

**CS54: Java Programming TERM: Aug - Dec 2019**

**Title of the Project: Student Details**

**PROJECT TEAM MEMBERS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl. No** | **Semester Section** | **USN** |  | **Name** |
| 1 | 5th C | 1MS17CS117 | Somandra Singh Rathore |  |
| 2 | 5th C | 1MS17CS138 | Yash Singh Chouhan |  |
| 3 | 5th C | 1MS17CS102 | Ruthwik Srikanth |  |

**M.S. RAMAIAH INSTITUTE OF TECHNOLOGY**

(Autonomous Institute, Affiliated to VTU)

**MICROSERVICES INTRODUCTION**

Microservices is an approach to software architecture that builds a large, complex application from multiple small components that each perform a single function, such as authentication, notification, or payment processing. Each microservice is a distinct unit within the software development project, with its own codebase, infrastructure, and database. The microservices work together, communicating through web APIs or messaging queues to respond to incoming events.

## **Advantages of Microservices**

* Developing a microservice does not require a big team.
* Each service of the application can be modified independently.
* Developing, modifying, and deploying a service does not affect the operation of another service.
* Does not require a long-term commitment to a certain type of technology.
* Allows flexible use of languages, frameworks, and databases within the same application
* Decreased risk attached to deployment, smoother operation, and fewer customer complaints.
* Easy to implement continuous deployment.
* Latest technologies can be adopted quickly
* Fault isolations: the crash of microservice does not make the entire application crash.
* Efficiently scalable:   
  - Only those services need to be scaled that are in demand.
* Easy to use with Docker (and other container technologies).
* Does not require big design efforts upfront.
* Enables flexible product evolution.
* Facilitates early release and continuous iteration.
* Allows quick time-to-market and Agile development.
* Easy to integrate with third party services and tools.

**Disadvantages of Microservices**

* Expensive remote calls (instead of in-process calls).
* Increased network latency.
* Increased processing time.
* Developing a use case could need the cooperation of multiple teams.
* Moving responsibilities between services are difficult and complex.
* Communication between multiple teams can cause decreased productivity.
* Duplication of efforts and replication of functionalities.
* Moving an engineer to new team could be problematic if they use a different stack.

## **Examples of Services based on Microservices**

* Netflix
* Amazon
* Twitter
* Paypal

# Problem Statement

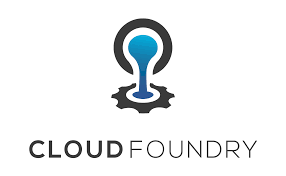
To develop a RESTFUL API micro service for a Student Details database using Spring Boot. And host the API on cloud foundry and also have a Web Application for the same. The micro service must provide appropriate API Endpoints to offer all the CRUD operation on the same database. Then the same API micro service must be hosted on cloud foundry and be functional.



# FUNCTIONAL REQUIREMENTS

1. Spring Tool Suite or Eclipse IDE
2. Latest version of Java Development Kit
3. SOAPUI
4. Cloud Foundry
5. MYSQL Work Bench



**Architecture**

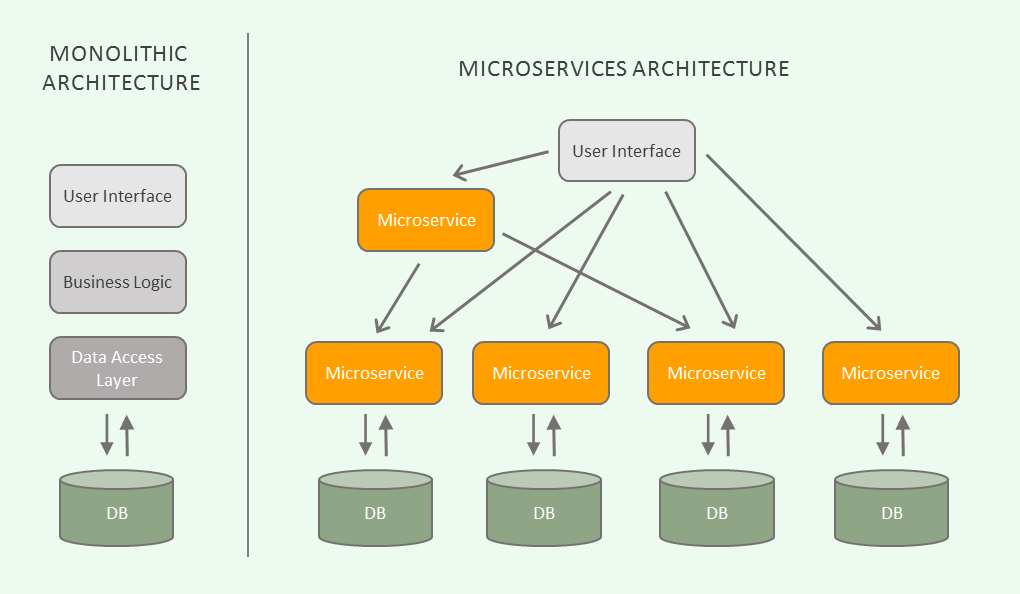
JSP Form User Interface

Controller Class

Service Class

Dao Implementation Class

Database



# API FLOW CHART

1.Create Module

**POST /enroll**

**CONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**JPA**

**REPOSI**

**TORY**



This module accepts Student details such as Student’s Name, DOB, Email and his/her Department and checks if this is his/her first attempt or not. It accepts all the subjects he/she registered. It will save the data automatically in the database with unique ID.

2.Retrieve module

**GET**

**/viewstudents**

**C**

**ONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**JPA**

**REPOSI**

**TORY**



This module retrieves all the Student details such as Student’s Name, DOB, Email and his/her Department, checks if this is his/her first attempt or not and all the registered subjects from the Database and display them.

3.Search By ID

**GET**

**/viewstudents1**

**/**

**{**

**id**

**}**

**CONTRO**

**LLER**

**DATA**

**ACCESSING**

**OBJECT**

**REPOSI**

**TORY**



This module accepts Student’s ID and searches for that ID in the database. if found then it returns the corresponding Student’s Name, DOB, Email and his/her Department, checks if this is his/her first attempt or not and all the registered subjects from the Database and display them.

4.Group By Dept

**POST**

**/viewstudents1/dept**

**CONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**REPOSI**

**TO**

**R**

**Y**



This module accepts Department Name and searches for that Name in the database. if found then it returns the corresponding Student’s Name, DOB, Email and his/her Department, checks if this is his/her first attempt or not and all the registered subjects from the Database and display them.

5.Group By Gender

**POST**

**/viewstudents2/groupbygender**

**CONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**REPOSI**

**TO**

**R**

**Y**



This module accepts Student’s Gender and searches for that Gender in the database. if found then it returns the corresponding Student’s Name, DOB, Email and his/her Department, checks if this is his/her first attempt or not and all the registered subjects from the Database and display them.

6.Delete Student By ID

**DELETE**

/deletestudent/{id}

**CONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**REPOSI**

**TO**

**R**

**Y**



This module accepts Student’s ID and searches for that ID in the database. if found then it deletes the corresponding record from the database. If the ID was not found in the database then it returns “Student” does not exist”.

7.Update

**POST**

**/editsave/{id}**

**CONTROLLER**

**DATA**

**ACCESSING**

**OBJECT**

**REPOSI**

**TORY**



This module accepts Student details such as Student’s Name, Department, Country and his/her Subjects. It updates the student’s details with the new values to the database and returns the same.

**IMPLEMENTATION DETAILS**

CRUD operations

* **Create an Entry in Database**

@RequestMapping(value ="/enroll",method = RequestMethod.***GET***)

**public** String newRegistration(ModelMap model) {

Student student = **new** Student();

model.addAttribute("student", student);

**return** "enroll";

}

@RequestMapping(value ="/save",method = RequestMethod.***POST***)

**public** String saveRegistration(@Valid Student student,

BindingResult result, ModelMap model,RedirectAttributes redirectAttributes) {

**if** (result.hasErrors()) {

**return** "enroll";//will redirect to viewemp request mapping

}

studentService.save(student);

//redirectAttributes.addFlashAttribute("message", "Student " + student.getFirstName()+" "+student.getLastName() + " saved");

**return** "redirect:/viewstudents/1";//will redirect to viewemp request mapping

}

/\*Function to insert data in Database\*/

**public** **void** setDataSource(DataSource dataSource) {

template = **new** JdbcTemplate(dataSource);

}

**public** **void** save(Student p) {

//String sql="insert into world.student(firstName,lastName,sex,dob,email,section,country,firstAttempt,subjects) values('"+p.getFirstName()+"','"+p.getLastName()+"','"+p.getSex()+"','"+p.getDob()+"','"+p.getEmail()+"','"+p.isFirstAttempt()+"','"+p.getCountry()+"',true,'"+convertListToDelimitedString(p.getSubjects())+"')";

String sql="insert into student(firstName,lastName,sex,dob,email,section,country,firstAttempt,subjects,department) values('"+p.getFirstName()+"','"+p.getLastName()+"','"+p.getSex()+"','"+ConvertDate(p.getDob())+"','"+p.getEmail()+"','"+p.getSection()+"','"+p.getCountry()+"',"+p.isFirstAttempt()+",'"+convertListToDelimitedString(p.getSubjects())+"','"+p.getDepartment()+"')";

System.***out***.println(sql);

template.update(sql);

}

• **Delete**

/\* It deletes record for the given id and redirects to /viewstudents \*/

@RequestMapping(value="/deletestudent/{id}",method = RequestMethod.***GET***)

**public** ModelAndView delete(@PathVariable **int** id){

studentService.delete(id);

**return** **new** ModelAndView("redirect:/viewstudents/1");

}

/\* It deletes record for the given id and redirects to /viewstudents \*/

@RequestMapping(value="/delete",method = RequestMethod.***GET***)

**public** ModelAndView delete(){

studentService.delete();

**return** **new** ModelAndView("redirect:/enroll");

}

/\*Delete by ID Function in DAO Implementation\*/

**public** **void** delete(**int** id) {

// **TODO** Auto-generated method stub

String sql="delete from Student where ID="+id+"";

template.update(sql);

}

/\*Delete ALL Function in DAO Implementation\*/

**public** **void** delete() {

// **TODO** Auto-generated method stub

String sql="delete from Student where ID>0";

template.update(sql);

}

* **Retrieve**

@RequestMapping("/viewstudents")

**public** ModelAndView viewstudents(){

List<Student> list=studentService.getAllStudents();

**return** **new** ModelAndView("viewstudents","list",list);

}

@RequestMapping(value="/viewstudents1",method=RequestMethod.***GET***)

**public** ModelAndView groupbydept(@Valid Student student,

BindingResult result, ModelMap model,RedirectAttributes redirectAttributes){

List<Student> list=studentService.getStudentByDept(student.getDepartment());

**return** **new** ModelAndView("viewstudents","list",list);

}

@RequestMapping(value="/viewstudents2",method=RequestMethod.***GET***)

**public** ModelAndView groupbysex(@Valid Student student,

BindingResult result, ModelMap model,RedirectAttributes redirectAttributes){

List<Student> list=studentService.getStudentByGen(student.getSex());

**return** **new** ModelAndView("viewstudents","list",list);

}

@RequestMapping(value ="/groupbygender",method = RequestMethod.***GET***)

**public** String newRegistration1(ModelMap model) {

Student student = **new** Student();

model.addAttribute("student", student);

**return** "groupbygender";

}

@RequestMapping(value ="/dept",method = RequestMethod.***GET***)

**public** String newRegistration2(ModelMap model) {

Student student = **new** Student();

model.addAttribute("student", student);

**return** "dept";

}

/\* It opens the student list for the given page id \*/

@RequestMapping(value="/viewstudents/{pageid}")

**public** ModelAndView edit(@PathVariable **int** pageid){

**int** total=2;

**if**(pageid==1){}

**else**{

pageid=(pageid-1)\*total+1;

}

List<Student> list=studentService.getStudentsByPage(pageid,total);

**return** **new** ModelAndView("viewstudents","list",list);

}

/\* It opens the record for the given id in edit student page \*/

@RequestMapping(value="/editstudent/{id}")

**public** String edit(@PathVariable **int** id,ModelMap model){

Student student=studentService.getStudentById(id);

model.addAttribute("student", student);

**return** "editstudent";

}

**public** List<Student> getAllStudents() {

**return** template.query("select \* from student",**new** ResultSetExtractor<List<Student>>(){

**public** List<Student> extractData(ResultSet rs) **throws** SQLException,

DataAccessException {

List<Student> list=**new** ArrayList<Student>();

**while**(rs.next()){

Student e=**new** Student();

e.setId(rs.getInt(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setSex(rs.getString(4));

e.setDob(rs.getDate(5));

e.setEmail(rs.getString(6));

e.setSection(rs.getString(7));

e.setCountry(rs.getString(8));

e.setFirstAttempt(rs.getBoolean(9));

e.setSubjects(*convertDelimitedStringToList*(rs.getString(10)));

e.setDepartment(rs.getString(11));

list.add(e);

}

**return** list;

}

});

}

**public** List<Student> getStudentsByPage(**int** pageid, **int** total) {

// **TODO** Auto-generated method stub

String sql="select \* from Student limit "+(pageid-1)+","+total;

**return** template.query(sql,**new** ResultSetExtractor<List<Student>>(){

**public** List<Student> extractData(ResultSet rs) **throws** SQLException,

DataAccessException {

List<Student> list=**new** ArrayList<Student>();

**while**(rs.next()){

Student e=**new** Student();

e.setId(rs.getInt(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setSex(rs.getString(4));

e.setDob(rs.getDate(5));

e.setEmail(rs.getString(6));

e.setSection(rs.getString(7));

e.setCountry(rs.getString(8));

e.setFirstAttempt(rs.getBoolean(9));

e.setSubjects(*convertDelimitedStringToList*(rs.getString(10)));

e.setDepartment(rs.getString(11));

list.add(e);

}

**return** list;

}

});

}

**public** Student getStudentById(**int** id) {

// **TODO** Auto-generated method stub

**return** template.query("select \* from student where ID="+id,**new** ResultSetExtractor<Student>(){

**public** Student extractData(ResultSet rs) **throws** SQLException,

DataAccessException {

Student e=**new** Student();

**while**(rs.next()){

e.setId(rs.getInt(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setSex(rs.getString(4));

e.setDob(rs.getDate(5));

e.setEmail(rs.getString(6));

e.setSection(rs.getString(7));

e.setCountry(rs.getString(8));

e.setFirstAttempt(rs.getBoolean(9));

e.setSubjects(*convertDelimitedStringToList*(rs.getString(10)));

e.setDepartment(rs.getString(11));

}

**return** e;

}

});

}

**public** List<Student> getStudentByDept(String department) {

// **TODO** Auto-generated method stub

**return** template.query("select \* from student where department="+"'"+department+"'",**new** ResultSetExtractor<List<Student>>(){

**public** List<Student> extractData(ResultSet rs) **throws** SQLException,

DataAccessException {

List<Student> list=**new** ArrayList<Student>();

**while**(rs.next()){

Student e=**new** Student();

e.setId(rs.getInt(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setSex(rs.getString(4));

e.setDob(rs.getDate(5));

e.setEmail(rs.getString(6));

e.setSection(rs.getString(7));

e.setCountry(rs.getString(8));

e.setFirstAttempt(rs.getBoolean(9));

e.setSubjects(*convertDelimitedStringToList*(rs.getString(10)));

e.setDepartment(rs.getString(11));

list.add(e);

}

**return** list;

}

});

}

**public** List<Student> getStudentByGen(String sex) {

// **TODO** Auto-generated method stub

**return** template.query("select \* from student where sex="+"'"+sex+"'",**new** ResultSetExtractor<List<Student>>(){

**public** List<Student> extractData(ResultSet rs) **throws** SQLException,

DataAccessException {

List<Student> list=**new** ArrayList<Student>();

**while**(rs.next()){

Student e=**new** Student();

e.setId(rs.getInt(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setSex(rs.getString(4));

e.setDob(rs.getDate(5));

e.setEmail(rs.getString(6));

e.setSection(rs.getString(7));

e.setCountry(rs.getString(8));

e.setFirstAttempt(rs.getBoolean(9));

e.setSubjects(*convertDelimitedStringToList*(rs.getString(10)));

e.setDepartment(rs.getString(11));

list.add(e);

}

**return** list;

}

});

}

* **Update**

/\* It opens the record for the given id in edit student page \*/

@RequestMapping(value="/editstudent/{id}")

**public** String edit(@PathVariable **int** id,ModelMap model){

Student student=studentService.getStudentById(id);

model.addAttribute("student", student);

**return** "editstudent";

}

/\* It updates record for the given id in edit student page and redirects to /view students \*/

@RequestMapping(value="/editsave",method = RequestMethod.***POST***)

**public** ModelAndView editsave(@ModelAttribute("student") Student emp){

System.***out***.println("id is"+emp.getId());

studentService.update(emp);

**return** **new** ModelAndView("redirect:/viewstudents/1");

}

**public** **void** update(Student p) {

String sql="update Student set firstName='"+p.getFirstName()+"',lastName='"+p.getLastName()+"',sex='"+p.getSex()+"',dob='"+ConvertDate(p.getDob())+"',email='"+p.getEmail()+"',section='"+p.getSection()+"' ,country='"+p.getCountry()+"', firstAttempt="+p.isFirstAttempt()+",subjects='"+convertListToDelimitedString(p.getSubjects())+"',department='"+p.getDepartment()+"' where ID="+p.getId();

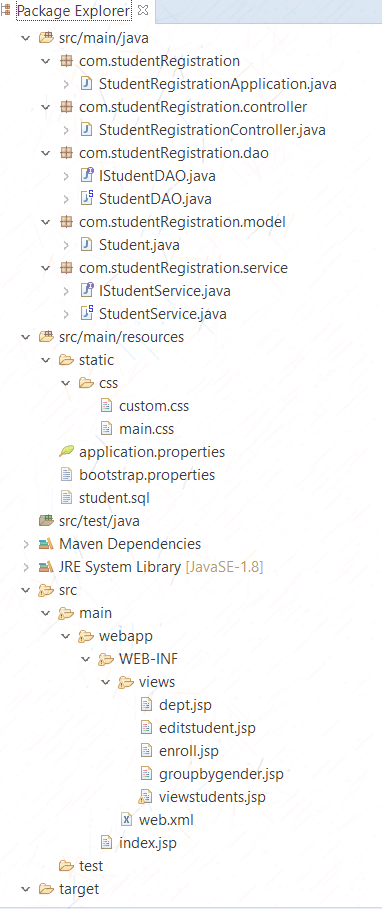
System.***out***.println(sql);

template.update(sql);

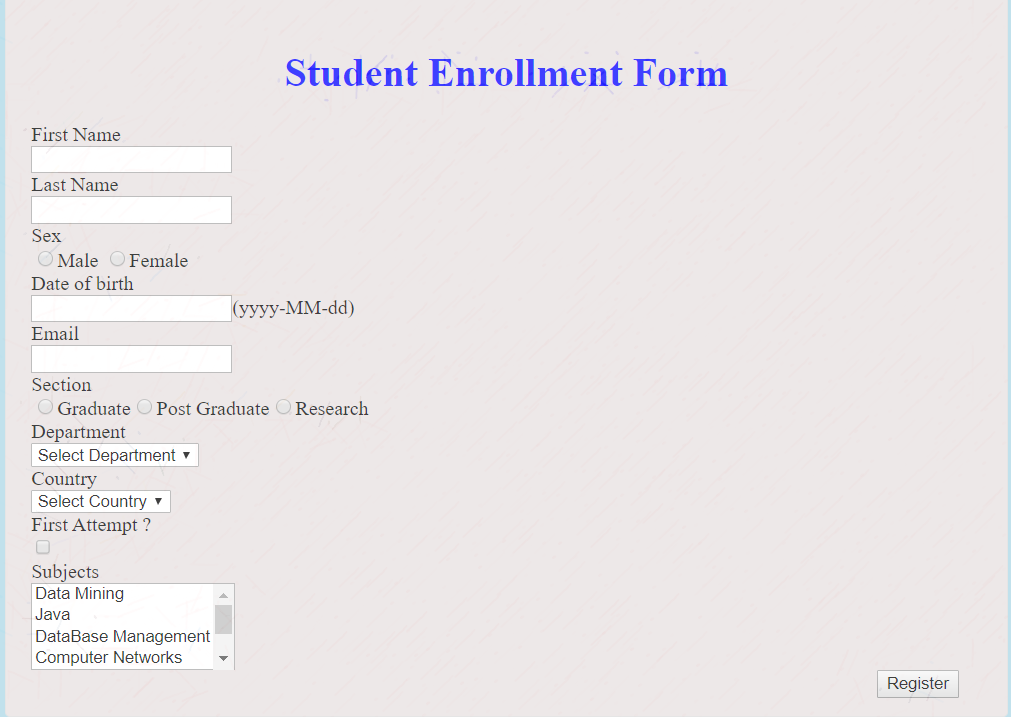
}

# Screen Shots

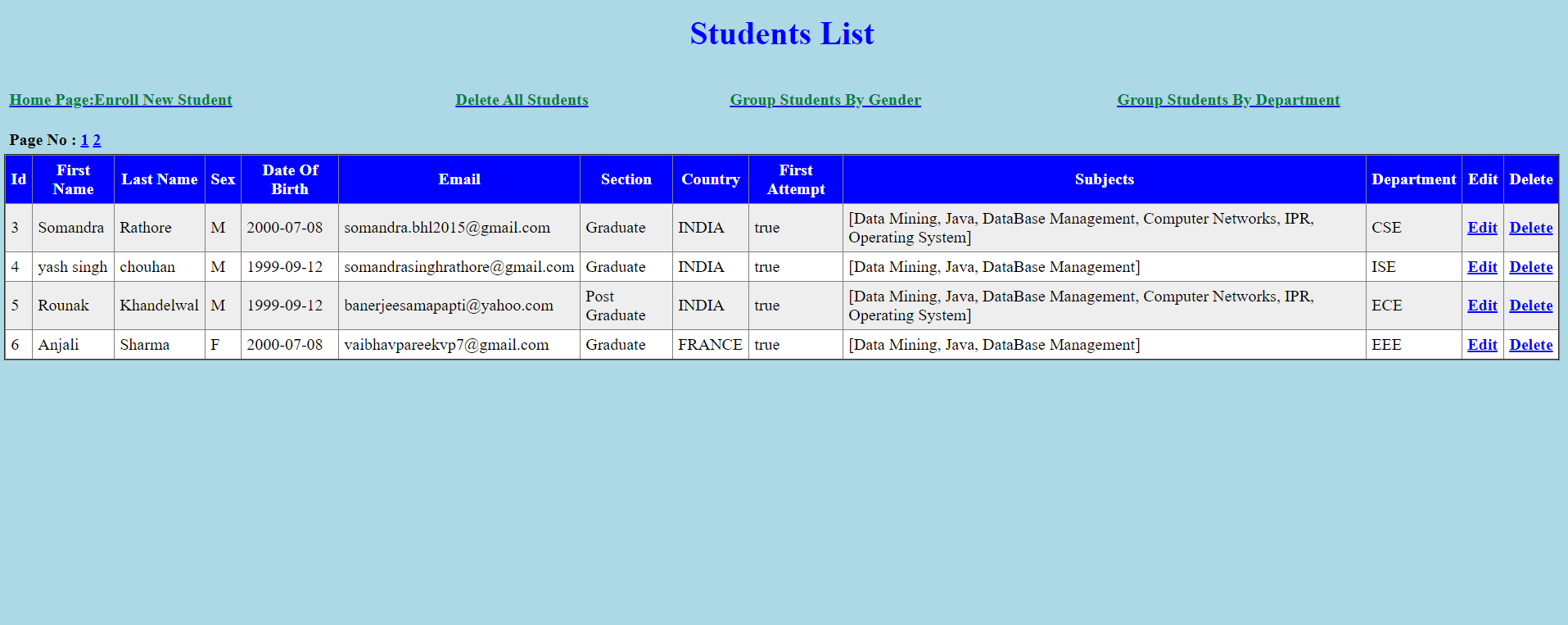
**Project Structure:**

****

**Insert Student:**



**Information About Students:**

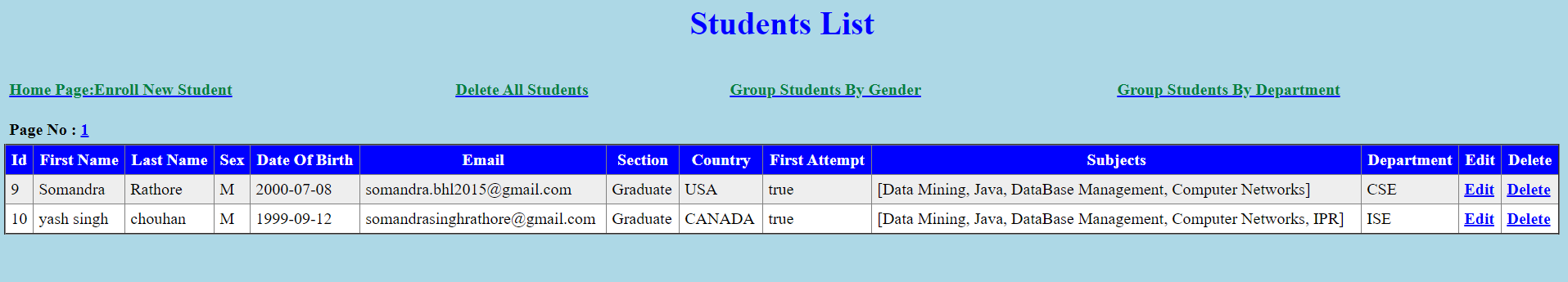


**Information of students group by Department:**

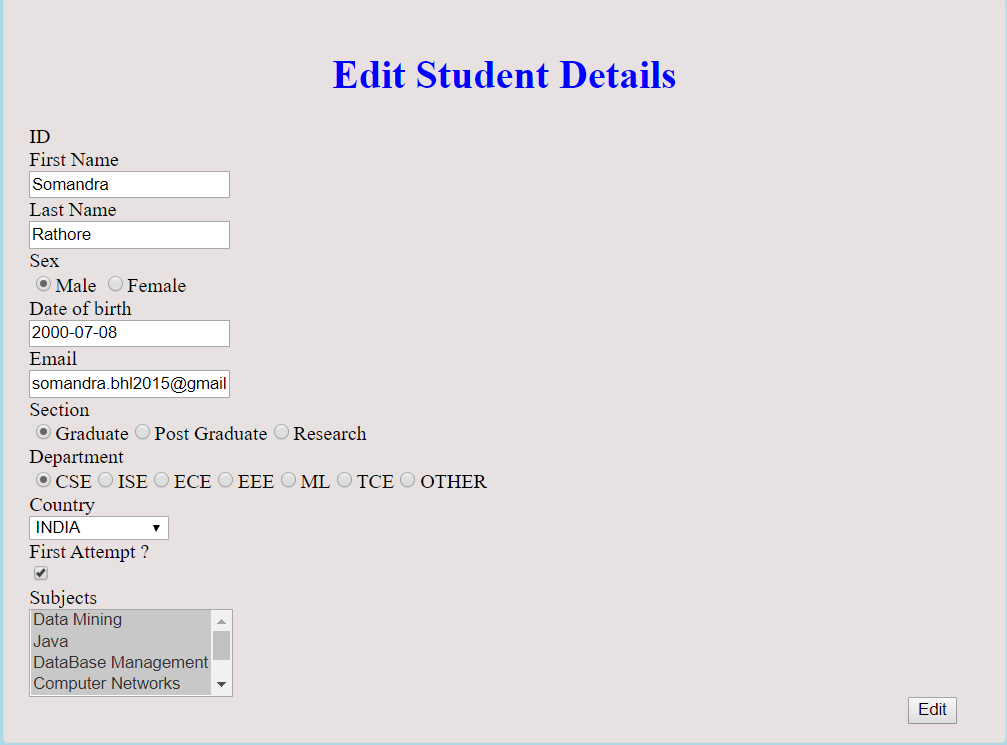


**Information of students group by Gender:**

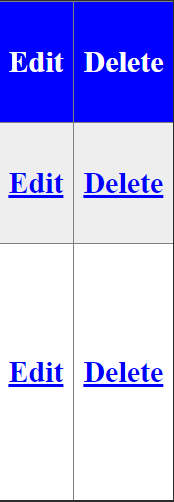
****

****

**Update:**



**Delete:**



****

**Read Me**

* The Spring Boot Application connects to the MySQL server using JPA Repository.
* JPA Repository offers abstraction for updating , deleting , inserting object into the database. We create a list of objects that gets updated in the database.
* The functions that can be used are : JPARepository.save() and JPARepository.get().
* This can be achieved by extending the repository to JPARepository class.
* src/main/resource/application.properties is used to establish Connection to Database.

/\*spring.datasource.url=jdbc:mysql://localhost:3306/studreg?autoReconnect=true&useSSL=false

spring.datasource.username=root

spring.datasource.password=pass123

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

spring.mvc.view.prefix:/WEB-INF/views/

spring.mvc.view.suffix:.jsp\*/

* To run the application run com.studentRegistration/StudentRegistrationApplication.java as Java Application.

/\*package com.studentRegistration;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.support.SpringBootServletInitializer;

@SpringBootApplication

public class StudentRegistrationApplication extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(org.springframework.boot.builder.SpringApplicationBuilder application) {

return application.sources(StudentRegistrationApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(StudentRegistrationApplication.class, args);

}

}

\*/

* Once Server Started open browser go to localhost:8080/enrol to initiate application.
* Data through Jsp forms are recieved by the application through several RequestMappings.
* StudentRegistrationController.java Class Handled All the page Requests and their corresponding mappings.
* The RESTful API is done by receiving a set of requests and then send appropriate data back to the request.
* The API is then compiled into a JAR file using maven, which adds only the dependencies that are used , and ignore the ones that are not specified.
* Since we have a MySQL Connection established , we need to provide the service in pivotal dashboard.
* This is done by enabling ClearMySQL Database service by giving an appropriate Instance Name.
* The API is then hosted on Cloud Foundry by pushing the JAR file to the pivotal dashboard.

//To login in Terminal Type cf then follow these commands

>> cf login -a api.run.pivotal.io

//To push application

>> cf push studentRegistration -p target\studentRegistration.jar

//To logout

>> cf logout