LIBRARY MANAGEMENT SYSTEM

Developed a web application named Library Management System with Angular for front end and spring boot for Restful backend, which is used to allow the user (i.e. admin) to access, manage, and view all the operations of the library i.e. adding new admins, registering new students, managing book operations i.e. adding books to library ,displaying the books available , issuing books to students and viewing the issued books .

**Aim & Objective**

The **aim** of the **e-library management** system is to provide remote educational services.

**PROJECT MODULES**

· **Dashboard**

· **Student registration,login**

· **Admin registration,login**

· **adding books to library**

· **viewing books**

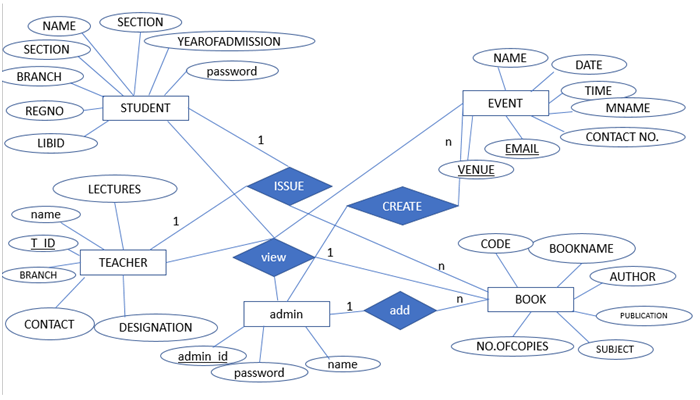
· **issuing Available books to only registered students**

The whole project is broadly divided into two categories

1. front end-with Angular.

2.Restful backend-Springboot.

ER DIAGRAM

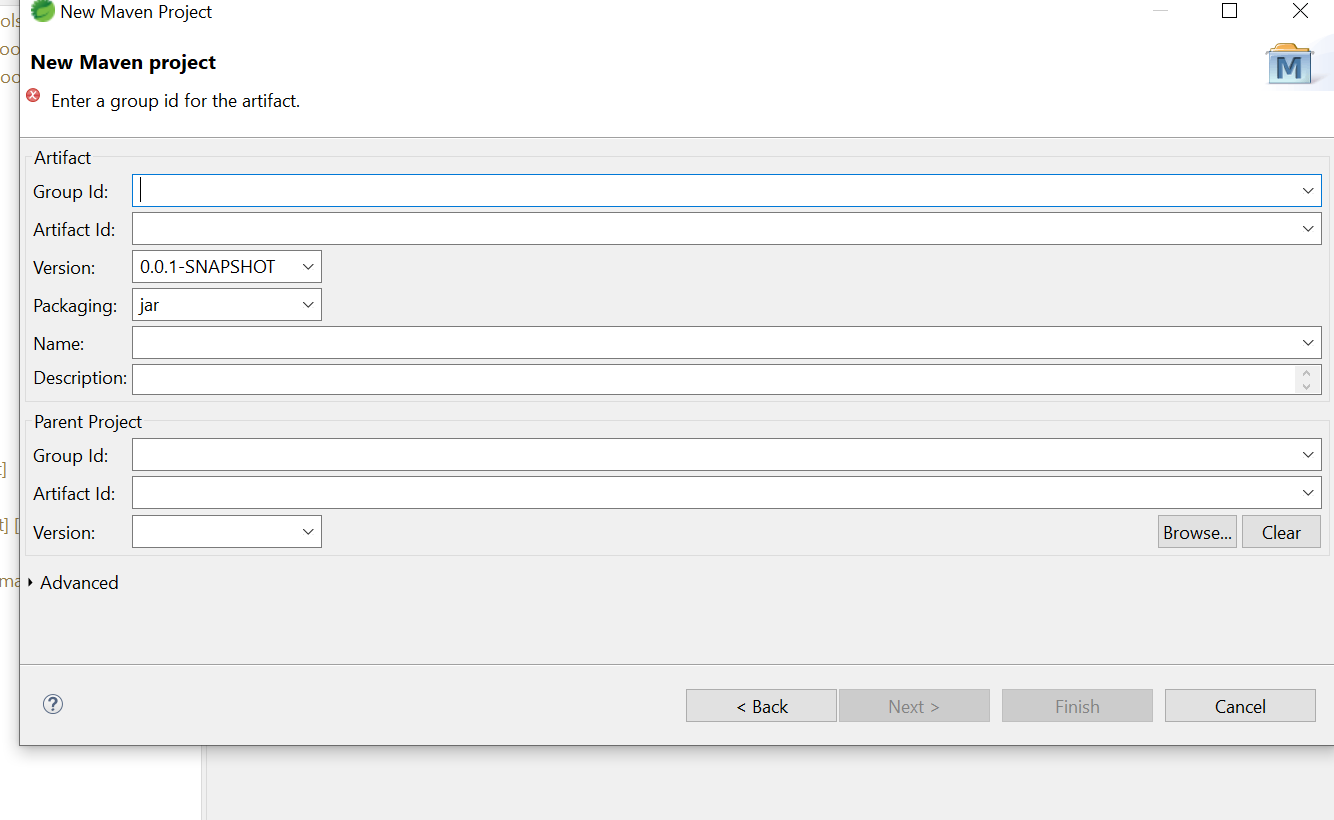


Basically, we will create two projects:

1. **AngularSpringLMS**: This project is used to develop CRUD RESTFul APIs for a simple **Library Management System** using Spring Boot , JPA and Oracle as a database.
2. **Angular App:** This project is used to develop application using Angular as front-end technology. This Angular application consumes CRUD Restful APIs developed and exposed by a  **AngularSpringLMS** project.

1. **The Spring Boot Application**

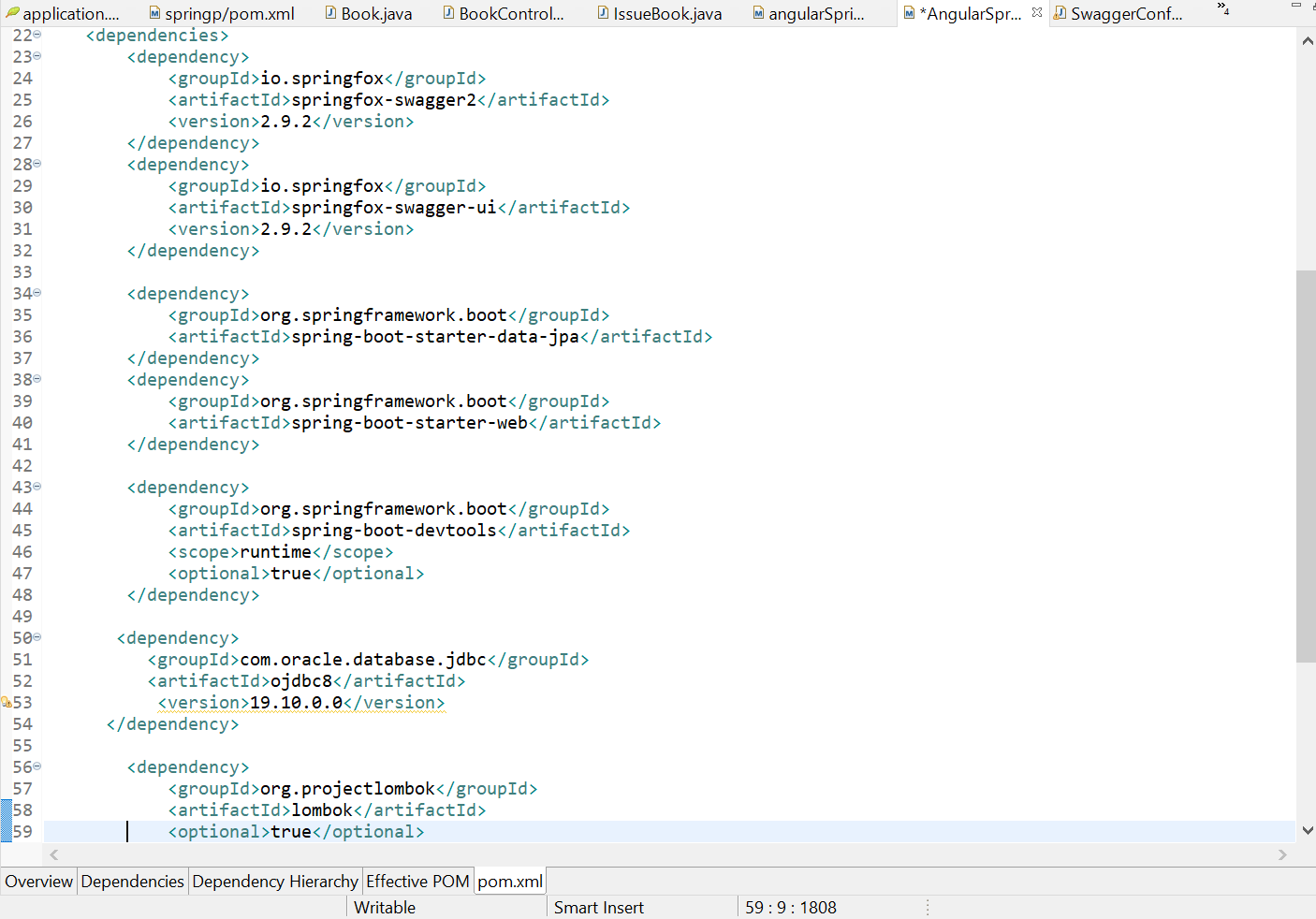
There are many ways in creating spring boot project.In sts bundle first we start with creating maven project



After giving groupId,artifact ID and packaging as jar click on next to create project.

## 2. The pom.xml File

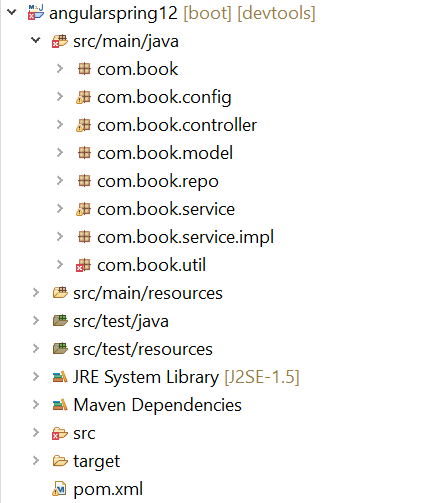
TO add dependencies required for the project we have to add dependencies in pom.xml file



we included [*spring-boot-starter-web*](https://search.maven.org/search?q=g:org.springframework.boot%20AND%20a:spring-boot-starter-web) because we'll use it for creating the REST service, and [*spring-boot-starter-jpa*](https://search.maven.org/search?q=g:org.springframework.boot%20AND%20a:spring-boot-starter-data-jpa) for implementing the persistence layer.

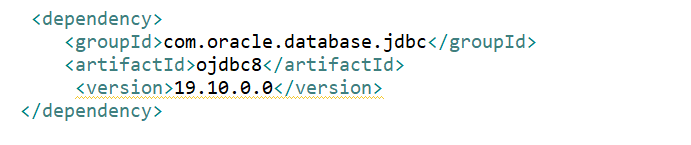
## 3.  Packaging Structure

For any project the structure will always be



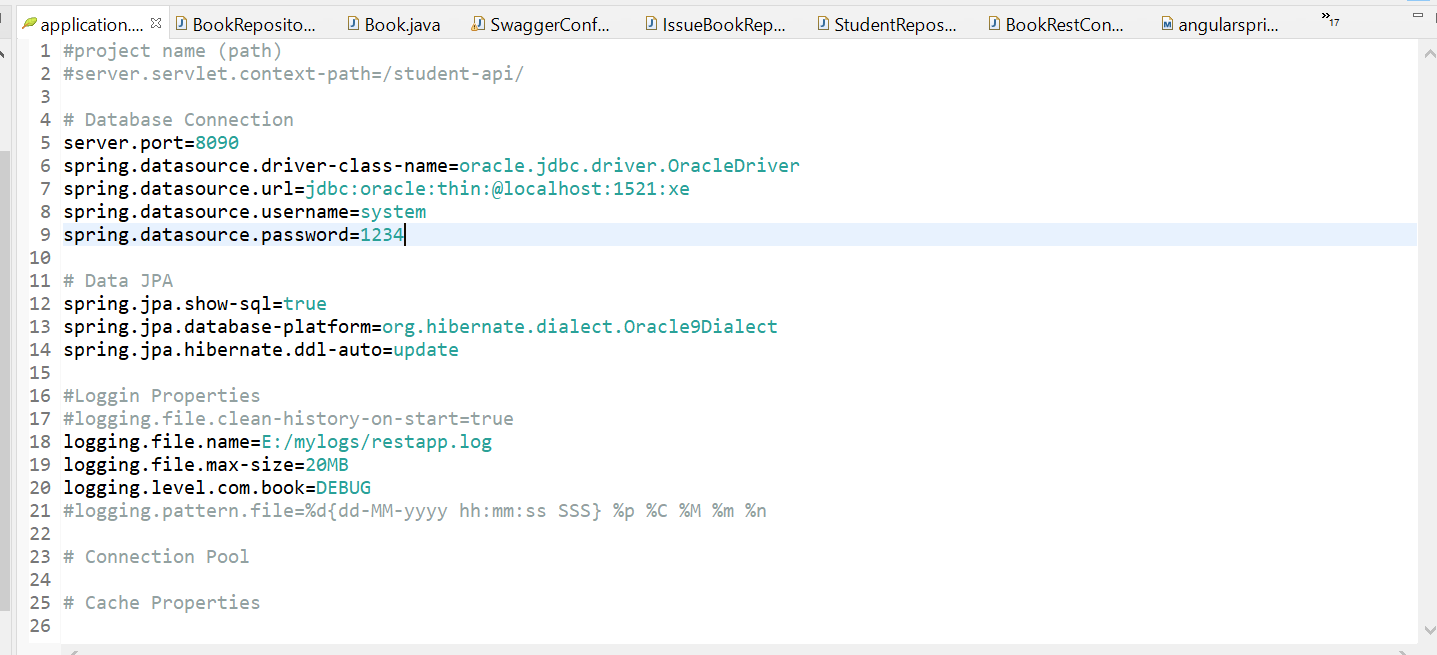
## 4. Configuring Database

In this project we used Oracle database .hence added that dependency in pom.xml file



 Configure *application.properties* to connect to your Oracle database. Add the following content to *application.properties* file:

Hibernate will automatically create database tables so you only need to manually create the database and configure an *application.properties* file.



## 5. Creating JPA Entity

## To quickly prototype our application's domain layer, we have created 5 JPA entity class, which will be responsible for modeling

## Student

## Addbook

## Issuebook

## User

## Sign up

* Add book

|  |
| --- |
| @Entity  public class Book {  @Id  @GeneratedValue  private Integer code;  private String book\_name;  private String author;  private String publication;  private String subject; |

* Student

|  |
| --- |
| @Entity  public class Student {  @Id  @GeneratedValue  private Integer id;  private String name;  private String course;  private String year;  private String email;  private String phone;  private String address;  } |

* Issue book

|  |
| --- |
| @Entity  public class IssueBook {  @Id  private int book\_id;  private int Student\_id;  private String issuedate;  private String returndate; |

* User

|  |
| --- |
| @Entity  **public** **class** Users {  @Id  @GeneratedValue    **private** Integer userId;  **private** String userName;  **private** String userPassword;  **private** BigInteger userPhone;  **private** String userEmail; |

## 

## SignUp

|  |
| --- |
| @Entity  **public** **class** signup {  @Id  @GeneratedValue  **private** **int** id;  **private** String name;  **private** String email;  **private** String contactno;  **private** String city;  **private** String state;  **private** String pwd; |

## Data Base Tables:

## AS we have used hibernate the tables are created automatically with the entities created.

## Hence as per given entities table are generated.

## Student table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Data Type** | **Length** | **Precision** | **Scale** | **Primary Key** | **Nullable** | **Default** | **Comment** |
| [STUDENT](javascript:ret_Column('SYSTEM.STUDENT');) | [ID](javascript:ret_Column('ID');) | Number | - | 10 | 0 | 1 | - | - | - |
|  | [ADDR](javascript:ret_Column('ADDR');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [COURSE](javascript:ret_Column('COURSE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [EMAIL](javascript:ret_Column('EMAIL');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [FEE](javascript:ret_Column('FEE');) | Float | 22 | 126 | - | - | nullable | - | - |
|  | [NAME](javascript:ret_Column('NAME');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [ADDRESS](javascript:ret_Column('ADDRESS');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [PHONE](javascript:ret_Column('PHONE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [YEAR](javascript:ret_Column('YEAR');) | Varchar2 | 1020 | - | - | - | nullable | - | - |

## Book table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Data Type** | **Length** | **Precision** | **Scale** | **Primary Key** | **Nullable** | **Default** | **Comment** |
| [BOOK](javascript:ret_Column('SYSTEM.BOOK');) | [CODE](javascript:ret_Column('CODE');) | Number | - | 10 | 0 | 1 | - | - | - |
|  | [AUTHOR](javascript:ret_Column('AUTHOR');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [BOOK\_NAME](javascript:ret_Column('BOOK_NAME');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [NO\_OF\_COPIES](javascript:ret_Column('NO_OF_COPIES');) | Number | - | 10 | 0 | - | nullable | - | - |
|  | [PUBLICATION](javascript:ret_Column('PUBLICATION');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [SUBJECT](javascript:ret_Column('SUBJECT');) | Varchar2 | 1020 | - | - | - | nullable | - | - |

## 3.Issue book table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Data Type** | **Length** | **Precision** | **Scale** | **Primary Key** | **Nullable** | **Default** | **Comment** |
| [ISSUE\_BOOK](javascript:ret_Column('SYSTEM.ISSUE_BOOK');) | [CODE](javascript:ret_Column('CODE');) | Number | - | 10 | 0 | 1 | - | - | - |
|  | [BOOK\_ID](javascript:ret_Column('BOOK_ID');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [BOOK\_NAME](javascript:ret_Column('BOOK_NAME');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [ISSUE\_DATE](javascript:ret_Column('ISSUE_DATE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [RETURN\_DATE](javascript:ret_Column('RETURN_DATE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [STUDENT\_ID](javascript:ret_Column('STUDENT_ID');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [ISSUEDATE](javascript:ret_Column('ISSUEDATE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [RETURNDATE](javascript:ret_Column('RETURNDATE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [ID](javascript:ret_Column('ID');) | Number | - | 10 | 0 | - | - | - | - |

## 4. Users table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Data Type** | **Length** | **Precision** | **Scale** | **Primary Key** | **Nullable** | **Default** | **Comment** |
| [USERS](javascript:ret_Column('SYSTEM.USERS');) | [USER\_ID](javascript:ret_Column('USER_ID');) | Number | - | 10 | 0 | 1 | - | - | - |
|  | [USER\_EMAIL](javascript:ret_Column('USER_EMAIL');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [USER\_NAME](javascript:ret_Column('USER_NAME');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [USER\_PASSWORD](javascript:ret_Column('USER_PASSWORD');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [USER\_PHONE](javascript:ret_Column('USER_PHONE');) | Number | - | 19 | 2 | - | nullable | - | - |

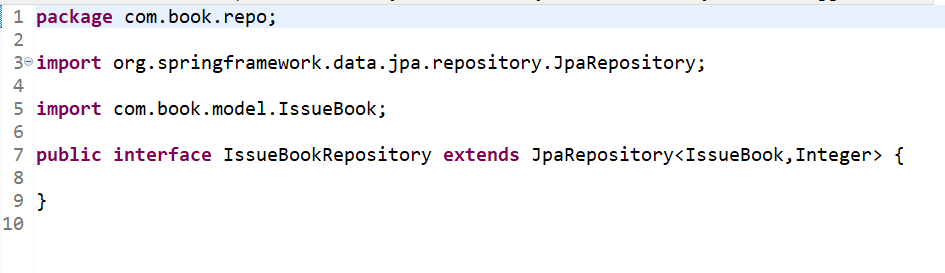
## 5.Signup table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Column** | **Data Type** | **Length** | **Precision** | **Scale** | **Primary Key** | **Nullable** | **Default** | **Comment** |
| [SIGNUP](javascript:ret_Column('SYSTEM.SIGNUP');) | [ID](javascript:ret_Column('ID');) | Number | - | 10 | 0 | 1 | - | - | - |
|  | [CITY](javascript:ret_Column('CITY');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [CONTACTNO](javascript:ret_Column('CONTACTNO');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [EMAIL](javascript:ret_Column('EMAIL');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [NAME](javascript:ret_Column('NAME');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [PWD](javascript:ret_Column('PWD');) | Varchar2 | 1020 | - | - | - | nullable | - | - |
|  | [STATE](javascript:ret_Column('STATE');) | Varchar2 | 1020 | - | - | - | nullable | - | - |

## 6. Creating a spring Repository

Since we'll need basic CRUD functionality on the *created* entities, we must also define a *Repository* interfaces:

Issue Book Repository Interface



Similarly we create all the other interface repositories.

## 

## 7.Creating Rest Controllers

For the models we have designed now we have to create controllers for

* issuing books,
* student,
* add books
* users.
* Sign up

STUDENT REST CONTROLLER

## package com.book.controller;

## 

## import java.util.List;

## import java.util.Optional;

## 

## import org.slf4j.Logger;

## import org.slf4j.LoggerFactory;

## import org.springframework.beans.factory.annotation.Autowired;

## import org.springframework.http.HttpStatus;

## import org.springframework.http.ResponseEntity;

## import org.springframework.web.bind.annotation.CrossOrigin;

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

## import org.springframework.web.bind.annotation.PathVariable;

## 

## import org.springframework.web.bind.annotation.PostMapping;

## import org.springframework.web.bind.annotation.RequestBody;

## import org.springframework.web.bind.annotation.RequestMapping;

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

## 

## import com.book.model.Student;

## import com.book.service.IStudentService;

## import com.book.util.StudentUtil;

## 

## 

## @RestController

## @RequestMapping("/rest/student")

## @CrossOrigin(origins = "http://localhost:4200")

## 

## public class StudentRestController {

## private Logger log = LoggerFactory.getLogger(BookRestController.class);

## 

## @Autowired

## private IStudentService service;

## // @Autowired

## // private StudentUtil util;

## @PostMapping("/save")

## public ResponseEntity<String> savestudent(

## @RequestBody Student student)

## {

## log.info("Entered into method with student data to save");

## 

## ResponseEntity<String> resp = null;

## try {

## 

## log.info("About to call save Operation");

## 

## Integer id = service.saveStudent(student);

## log.debug("student saved with id "+id);

## 

## String body = "Student '"+id+"' created";

## 

## resp = new ResponseEntity<String>(

## body,

## HttpStatus.CREATED); //201

## 

## log.info("Sucess response constructed");

## } catch (Exception e) {

## log.error("Unable to save Student : problem is :"+e.getMessage());

## resp = new ResponseEntity<String>(

## "Unable to Create Student",

## HttpStatus.INTERNAL\_SERVER\_ERROR); //500

## e.printStackTrace();

## }

## 

## 

## log.info("About to Exist save method with Response");

## return resp;

## }

## 

## @GetMapping("/all")

## public ResponseEntity<?> getAllStudents() {

## log.info("Entered into method to fetch students data");

## ResponseEntity<?> resp = null ;

## try {

## 

## log.info("About to call fetch students service");

## List<Student> list = service.getAllStudents();

## if(list!=null && !list.isEmpty()) {

## log.info("Data is not empty=>"+list.size());

## list.sort((b1,b2)->b1.getName().compareTo(b2.getName()));

## /\* JDK 1.8

## list = list.stream()

## .sorted((s1,s2)->s1.getName().compareTo(s2.getName()))

## .collect(Collectors.toList());

## \*/

## resp = new ResponseEntity<List<Student>>(list, HttpStatus.OK);

## } else {

## log.info("No Student exist: size "+list.size());

## 

## //resp = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

## resp = new ResponseEntity<String>(

## "No Students Found",

## HttpStatus.OK);

## }

## } catch (Exception e) {

## log.error("Unable to fetch student : problem is :"+e.getMessage());

## 

## resp = new ResponseEntity<String>(

## "Unable to Fetch Books",

## HttpStatus.INTERNAL\_SERVER\_ERROR); //500

## e.printStackTrace();

## }

## 

## log.info("About to Exist fetch all method with Response");

## return resp;

## }

## 

## 

## @GetMapping("/one/{id}")

## public ResponseEntity<?> getOneBook(

## @PathVariable Integer id

## )

## {

## log.info("Entered into Get one Student method");

## ResponseEntity<?> resp = null;

## try {

## log.info("About to make service call to fetch one record");

## Optional<Student> opt = service.getOneStudent(id);

## if(opt.isPresent()) {

## log.info("student exist=>"+id);

## resp = new ResponseEntity<Student>(opt.get(), HttpStatus.OK);

## //resp = ResponseEntity.ok(opt.get());

## } else {

## log.warn("Given student id not exist=>"+id);

## resp = new ResponseEntity<String>(

## "student '"+id+"' not exist",

## HttpStatus.BAD\_REQUEST);

## }

## } catch (Exception e) {

## log.error("Unable to process request fetch " + e.getMessage());

## 

## resp = new ResponseEntity<String>(

## "Unable to process student fetch",

## HttpStatus.INTERNAL\_SERVER\_ERROR);

## e.printStackTrace();

## }

## 

## return resp;

## }

## 

## }

## 

## 

## the annotation enables Cross-Origin Resource Sharing (CORS) on the server. This step isn't always necessary. Since we are deploying our Angular frontend to <http://localhost:4200> and our Boot backend to <http://localhost:8090> , the browser would otherwise deny requests from one to the other.

**Issue Book Controller**

package com.book.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

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

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

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

import com.book.model.IssueBook;

import com.book.service.IIssueBookService;

@RestController

@RequestMapping("/rest/issuebook")

@CrossOrigin(origins = "http://localhost:4200")

public class IssueBookRestController {

private Logger log = LoggerFactory.getLogger(BookRestController.class);

@Autowired

private IIssueBookService service;

// @Autowired

// private StudentUtil util;

@PostMapping("/save")

public ResponseEntity<String> saveissuebook(

@RequestBody IssueBook issueBook)

{

log.info("Entered into method with issuebook data to save");

ResponseEntity<String> resp = null;

try {

log.info("About to call save Operation");

Integer id = service.saveIssuebook(issueBook);

log.debug("issuebook saved with id "+id);

String body = "issuebook '"+id+"' created";

resp = new ResponseEntity<String>(

body,

HttpStatus.CREATED); //201

log.info("Sucess response constructed");

} catch (Exception e) {

log.error("Unable to save issuedbook : problem is :"+e.getMessage());

resp = new ResponseEntity<String>(

"Unable to Create issuebook",

HttpStatus.INTERNAL\_SERVER\_ERROR); //500

e.printStackTrace();

}

log.info("About to Exist save method with Response");

return resp;

}

@GetMapping("/all")

public ResponseEntity<?> getAllStudents() {

log.info("Entered into method to fetch issuebooks data");

ResponseEntity<?> resp = null ;

try {

log.info("About to call fetch issuebook service");

List<IssueBook> list = service.getAllIssuebooks();

if(list!=null && !list.isEmpty()) {

log.info("Data is not empty=>"+list.size());

/\* JDK 1.8

list = list.stream()

.sorted((s1,s2)->s1.getName().compareTo(s2.getName()))

.collect(Collectors.toList());

\*/

resp = new ResponseEntity<List<IssueBook>>(list, HttpStatus.OK);

} else {

log.info("No Issuebook exist: size "+list.size());

//resp = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

resp = new ResponseEntity<String>(

"No issuebook Found",

HttpStatus.OK);

}

} catch (Exception e) {

log.error("Unable to fetch issuebook : problem is :"+e.getMessage());

resp = new ResponseEntity<String>(

"Unable to Fetch issuebooks",

HttpStatus.INTERNAL\_SERVER\_ERROR); //500

e.printStackTrace();

}

log.info("About to Exist fetch all method with Response");

return resp;

}

}

**ADD Book controller:**

package com.book.controller;

import java.util.List;

import java.util.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.DeleteMapping;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

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

import com.book.model.Book;

import com.book.util.BookUtil;

import com.book.service.IBookService;

@RestController

@RequestMapping("/rest/book")

@CrossOrigin(origins = "http://localhost:4200")

public class BookRestController {

private Logger log = LoggerFactory.getLogger(BookRestController.class);

@Autowired

private IBookService service;

@Autowired

private BookUtil util;

/\*\*

\* 1. Read JSON(Student) and convert to Object Format

\* Store data in Database. Return one Message.

\*/

@PostMapping("/save")

public ResponseEntity<String> saveBook(

@RequestBody Book book)

{

log.info("Entered into method with Book data to save");

ResponseEntity<String> resp = null;

try {

log.info("About to call save Operation");

Integer id = service.saveBook(book);

log.debug("Book saved with id "+id);

String body = "Book '"+id+"' created";

resp = new ResponseEntity<String>(

body,

HttpStatus.CREATED); //201

log.info("Sucess response constructed");

} catch (Exception e) {

log.error("Unable to save book : problem is :"+e.getMessage());

resp = new ResponseEntity<String>(

"Unable to Create Book",

HttpStatus.INTERNAL\_SERVER\_ERROR); //500

e.printStackTrace();

}

log.info("About to Exist save method with Response");

return resp;

}

/\*\*

\* 2. Fetch all rows from database using Service

\* Sort data using name, return as JSON,

\* else String message no data found.

\*

\*/

@GetMapping("/all")

public ResponseEntity<?> getAllBooks() {

log.info("Entered into method to fetch Books data");

ResponseEntity<?> resp = null ;

try {

log.info("About to call fetch book service");

List<Book> list = service.getAllBooks();

if(list!=null && !list.isEmpty()) {

log.info("Data is not empty=>"+list.size());

list.sort((b1,b2)->b1.getBook\_name().compareTo(b2.getBook\_name()));

/\* JDK 1.8

list = list.stream()

.sorted((s1,s2)->s1.getName().compareTo(s2.getName()))

.collect(Collectors.toList());

\*/

resp = new ResponseEntity<List<Book>>(list, HttpStatus.OK);

} else {

log.info("No Book exist: size "+list.size());

//resp = new ResponseEntity<>(HttpStatus.NO\_CONTENT);

resp = new ResponseEntity<String>(

"No Students Found",

HttpStatus.OK);

}

} catch (Exception e) {

log.error("Unable to fetch books : problem is :"+e.getMessage());

resp = new ResponseEntity<String>(

"Unable to Fetch Books",

HttpStatus.INTERNAL\_SERVER\_ERROR); //500

e.printStackTrace();

}

log.info("About to Exist fetch all method with Response");

return resp;

}

/\*\*\*

\* 3. Get one student object based on ID (PathVariable).

\* If Object exist then return Student object

\* else provide message(String).

\*/

@GetMapping("/one/{id}")

public ResponseEntity<?> getOneBook(

@PathVariable Integer id

)

{

log.info("Entered into Get one Student method");

ResponseEntity<?> resp = null;

try {

log.info("About to make service call to fetch one record");

Optional<Book> opt = service.getOneBook(id);

if(opt.isPresent()) {

log.info("Book exist=>"+id);

resp = new ResponseEntity<Book>(opt.get(), HttpStatus.OK);

//resp = ResponseEntity.ok(opt.get());

} else {

log.warn("Given Book id not exist=>"+id);

resp = new ResponseEntity<String>(

"Book '"+id+"' not exist",

HttpStatus.BAD\_REQUEST);

}

} catch (Exception e) {

log.error("Unable to process request fetch " + e.getMessage());

resp = new ResponseEntity<String>(

"Unable to process student fetch",

HttpStatus.INTERNAL\_SERVER\_ERROR);

e.printStackTrace();

}

return resp;

}

/\*\*

\* 4. delete one row based on id (PathVariable)

\* First check given row exist or not?

\* if exist then call delete operation

\* else return NO RECORD MESSAGE

\*

\*/

@DeleteMapping("/remove/{id}")

public ResponseEntity<String> removeBook(

@PathVariable Integer id

)

{

log.info("Entered into delete method");

ResponseEntity<String> resp = null;

try {

log.info("About to make service call for data check");

boolean exist = service.isBookExist(id);

if(exist) {

service.deleteBook(id);

log.info("Book exist with given id and deleted=>"+id);

resp = new ResponseEntity<String>(

"Book '"+id+"' deleted",

HttpStatus.OK);

} else {

log.warn("Given Book id not exist =>"+id);

resp = new ResponseEntity<String>(

"Book '"+id+"' not exist",

HttpStatus.BAD\_REQUEST);

}

} catch (Exception e) {

log.error("Unable to perform Delete Operation =>" + e.getMessage());

resp = new ResponseEntity<String>(

"Unable to delete",

HttpStatus.INTERNAL\_SERVER\_ERROR);

e.printStackTrace()

}

return resp;

}

/\*\*

\* 5. Update

\*/

@PutMapping("/modify/{id}")

public ResponseEntity<String> updateBook(

@PathVariable Integer id,

@RequestBody Book book

)

{

log.info("Entered into Update method");

ResponseEntity<String> resp =null;

try {

log.info("About to check given id exist or not db");

Optional<Book> opt = service.getOneBook(id);

if(opt.isPresent()) {

log.info("Student present in database");

Book actual = opt.get();

util.mapToActualObject(actual,book);

service.updateBook(actual);

resp = new ResponseEntity<String>(

"Book '"+id+"' Updated",

//HttpStatus.RESET\_CONTENT

HttpStatus.OK

);

log.info("Book update done successfully");

} else {

log.info("Book not exist=>"+id);

resp = new ResponseEntity<String>(

"Book '"+id+"' not found",

//HttpStatus.RESET\_CONTENT

HttpStatus.BAD\_REQUEST

);

}

}

catch (Exception e) {

log.error("Unable to perform Update Operation=>" + e.getMessage() );

resp = new ResponseEntity<String>(

"Unable to process Update",

HttpStatus.INTERNAL\_SERVER\_ERROR);

e.printStackTrace();

}

return resp;

}

}

Similarly we will create the other user controller for implementing services of the model entities.

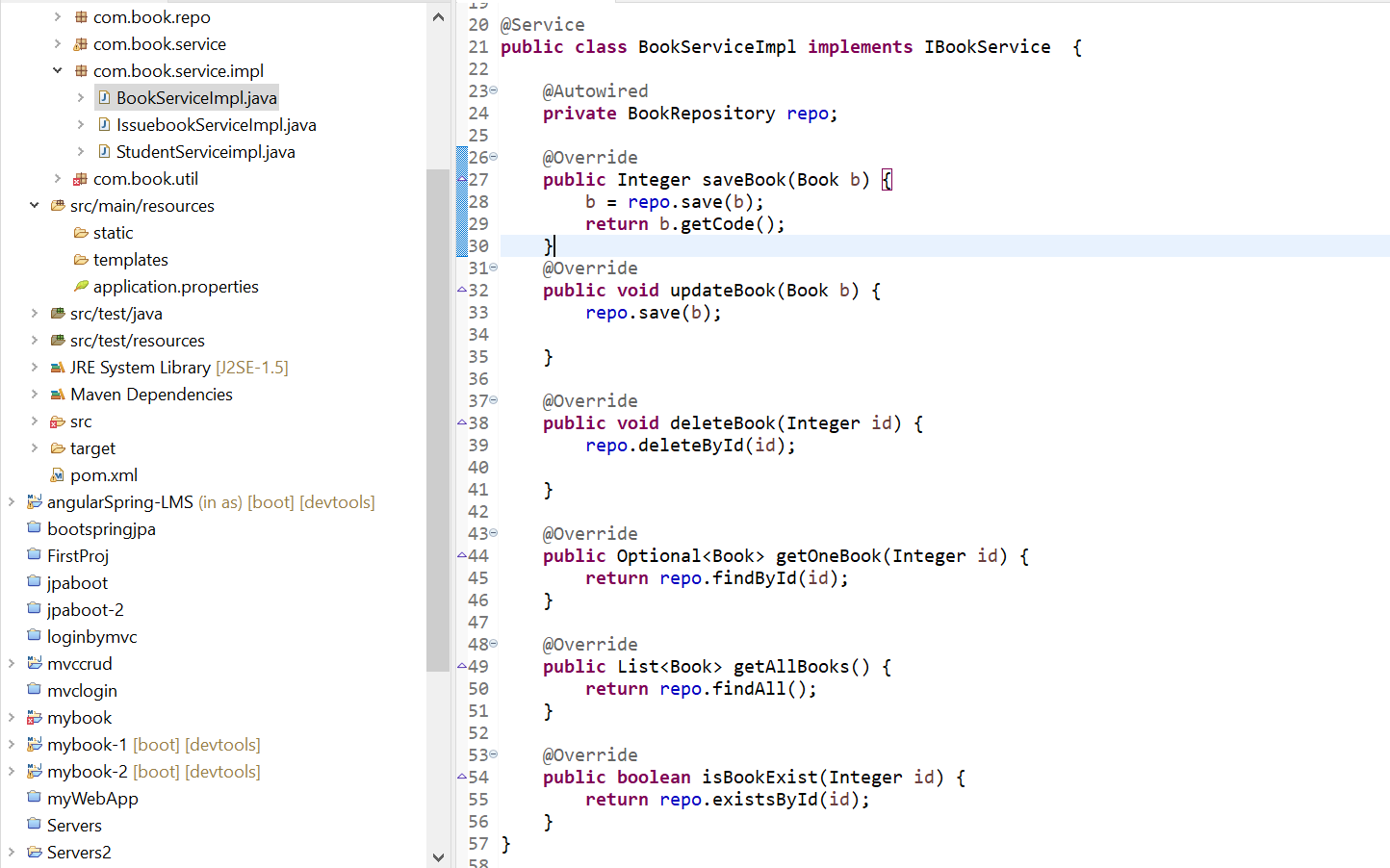
**8.Adding service layer**

Similar to repository interface we will create service interfaces i.e.



**9.Adding service Implementation**

a similar one to controller a service implementation layer is created for all the service interfaces



1. **Angular App**

With our Spring Boot application up and running, let's now create a Angular application, capable of consuming the REST controllers API.

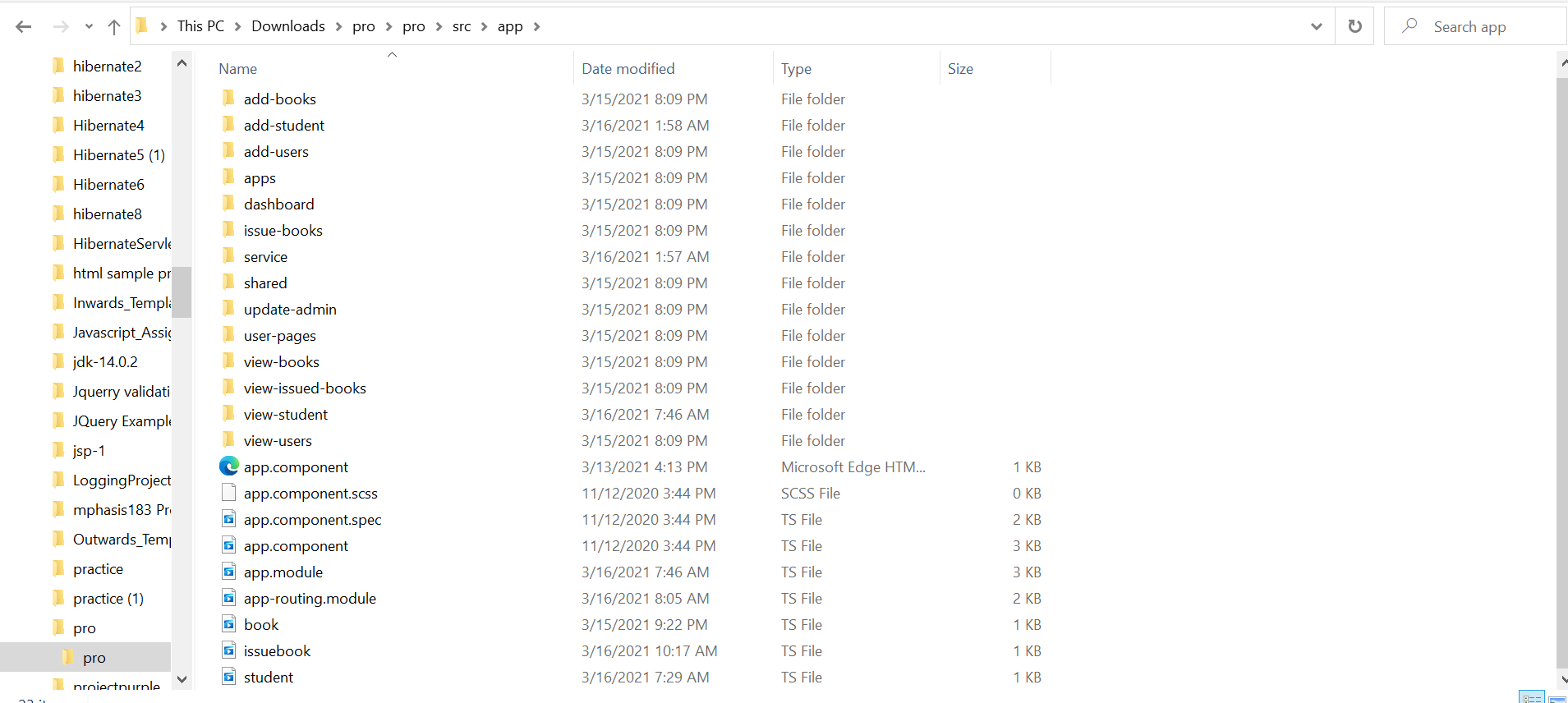
**1.creating a angular project**

In the command line give the command npm project to create a project

and then npm install to create node module .

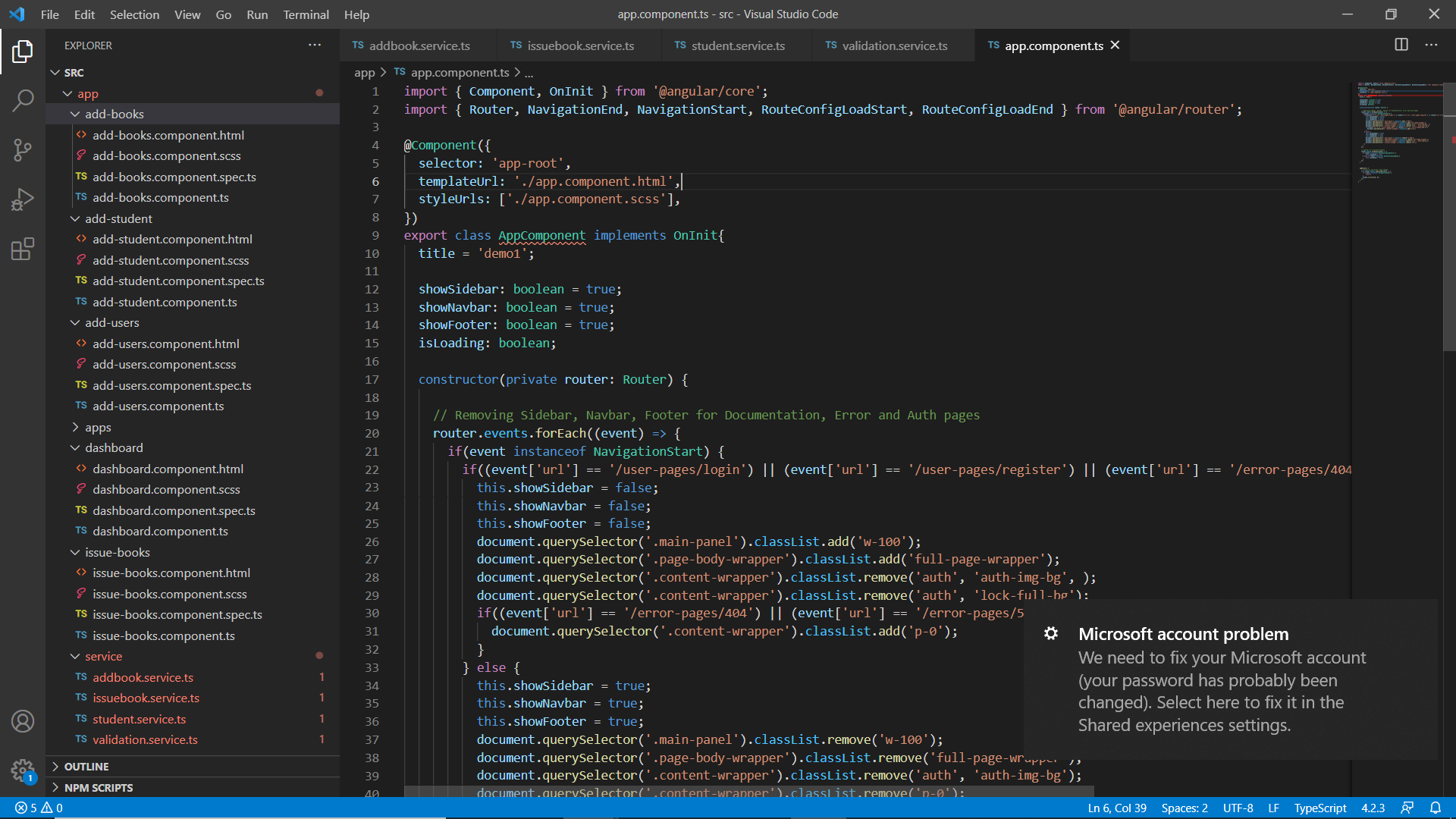
**2.Angular Folder Structure**

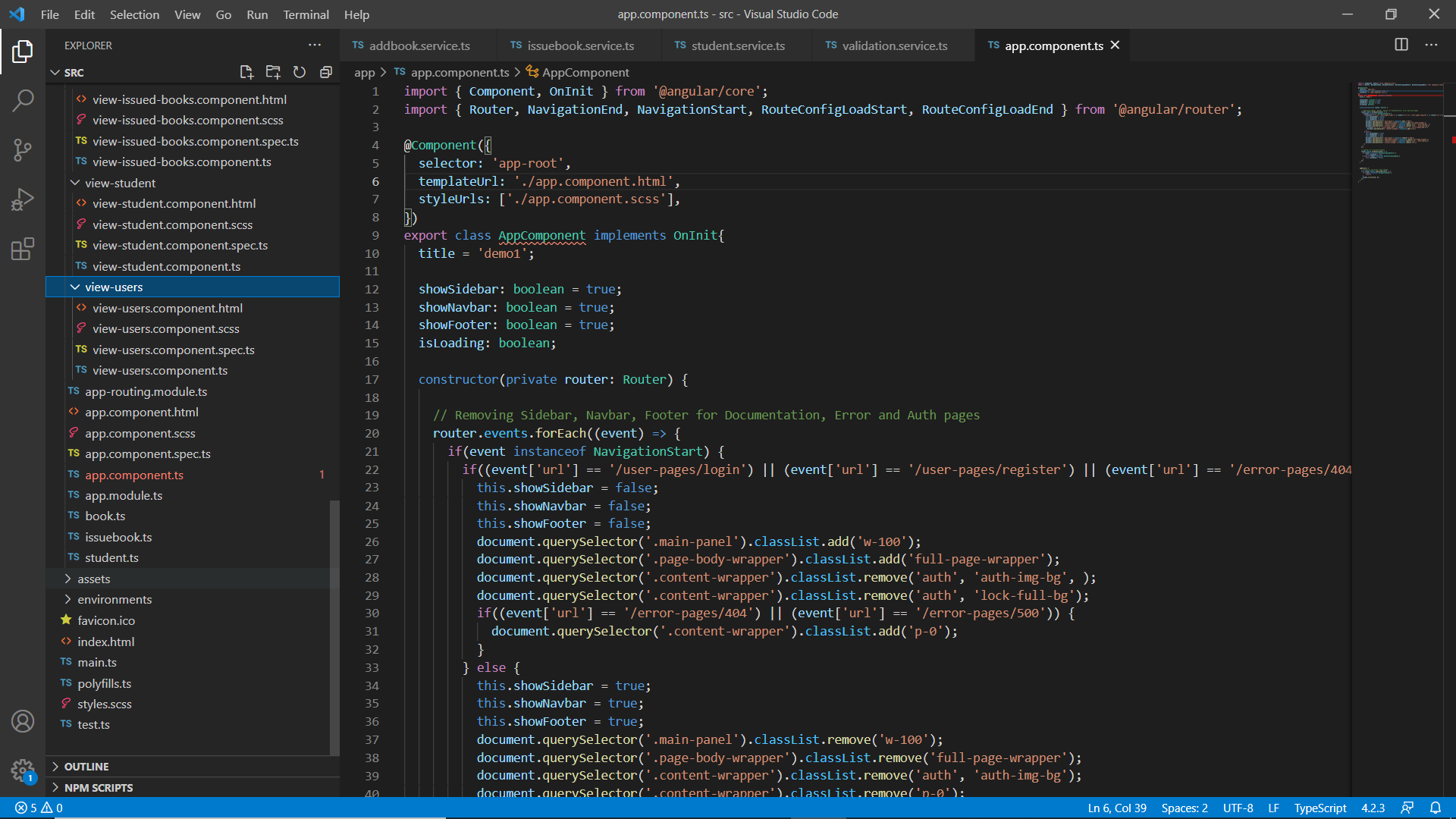
**BY giving ng g c folder name create all folders.**

****

**For the entities stored in the database to display it in the front end we have to create the components and configure them in app module,component.ts.**

**hence we have created all the components as shown below..**

****

****

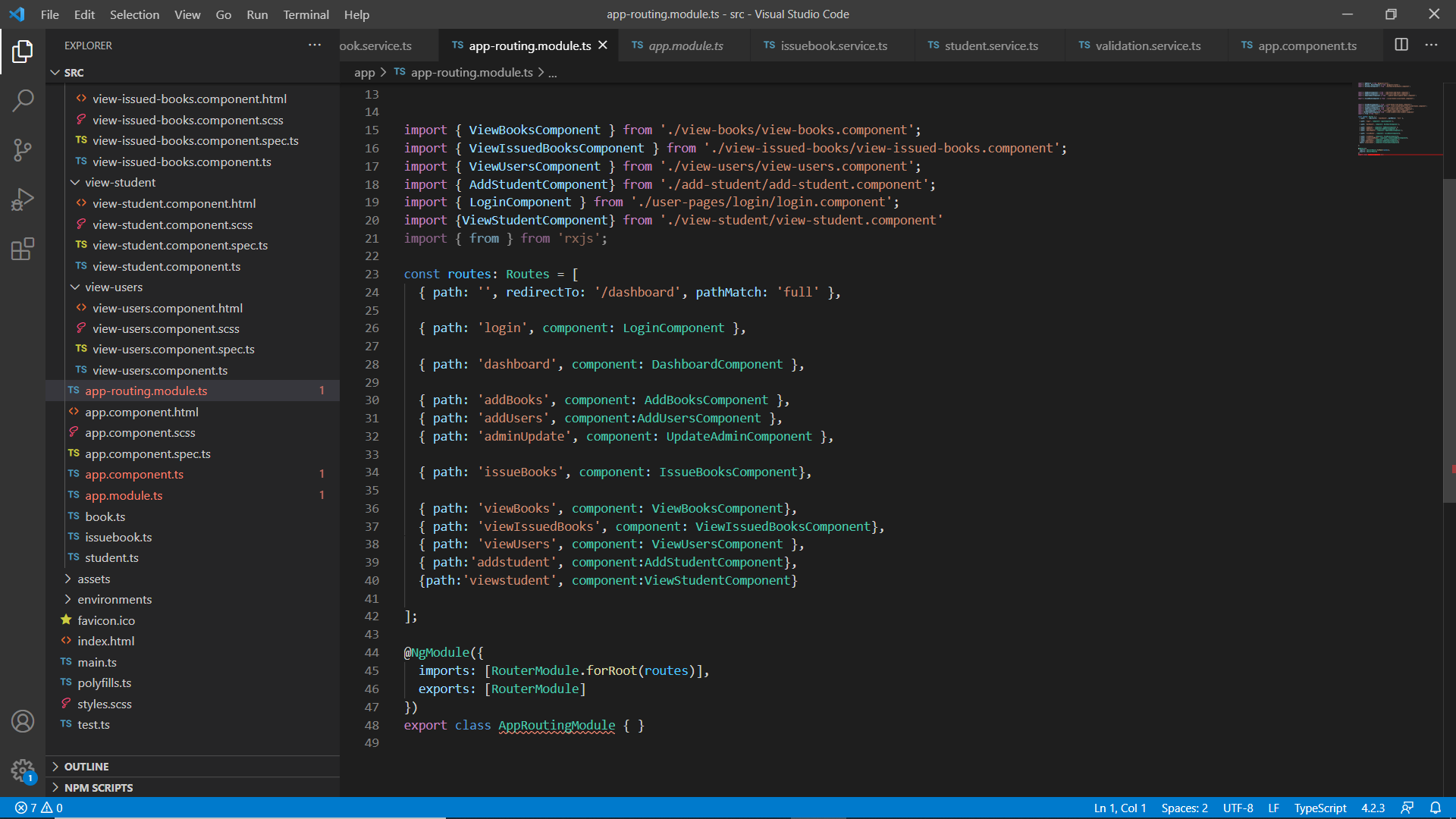
**3.App routing module.ts**

all the routing redirects are provided in this module.Although the components are functional in isolation, we still need to use a mechanism for calling them when the user clicks the buttons in the navigation bar.

This is where the [*RouterModule*](https://angular.io/api/router/RouterModule) comes into play. So, let's open the *app-routing.module.ts* file, and configure the module, so it can dispatch requests to the matching components:

A route is composed of two parts:

1. *Path –*  a *string* that matches the URL in the browser address bar
2. *Component* – the component to create when the route is active (navigated)



**4.service.ts**

With our client-side domain *all the entities* class already set, let's now implement a service class that performs GET and POST requests to the [http://localhost:8090/](http://localhost:8080/users)endpoint.

This will allow us to encapsulate access to the REST controllers in a single class, which we can reuse throughout the entire application.

the *Service* class is the thin middle-tier between the REST service and the application's presentation layer. Therefore, we need to define a component responsible for rendering the list of entities persisted in the database.

As we have 4 controller classes we need to write 4 service.ts files.

**Issue book service.ts**

**import { Injectable } from '@angular/core';**

**import { HttpClient } from '@angular/common/http';**

**import {Issuebook} from '../issuebook'**

**import { Observable } from 'rxjs';**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class NgserviceService {**

**private basePath = 'http://localhost:8090/rest/issuebook';**

**constructor(private \_http:HttpClient) { }**

**//issue book service**

**deleteOneIssuedBook(id: number): Observable<any> {**

**return this.\_http.delete(`${this.basePath}/remove/${id}`, {responseType: 'text'});**

**}**

**issueBook(issuebook: Issuebook): Observable<any> {**

**return this.\_http.post(`${this.basePath}/save`, issuebook , {responseType: 'text'});**

**}**

**getOneIssuedBook(id: number): Observable<Issuebook> {**

**return this.\_http.get<Issuebook>(`${this.basePath}/one/${id}`);**

**}**

**}**

**ADD BOOK Service.ts**

**export class NgserviceService {**

**private basePath = 'http://localhost:8090/rest/book';**

**private basePath1 = 'http://localhost:8090/rest/issuebook';**

**constructor(private \_http:HttpClient) { }**

**fetchBookListFromRemote():Observable<Book[]>{**

**return this.\_http.get<Book[]>(`${this.basePath}/all`);**

**}**

**addbook(book : Book):Observable<any>{**

**return this.\_http.post(`${this.basePath}/savebook`, book, {responseType: 'text'});**

**}**

**deleteOneBook(code: number): Observable<any> {**

**return this.\_http.delete(`${this.basePath}/remove/${code}`, {responseType: 'text'});**

**}**

**getOneBook(code: number): Observable<Book> {**

**return this.\_http.get<Book>(`${this.basePath}/one/${code}`);**

**}**

**updateBook(code: number, book: Book): Observable<any> {**

**return this.\_http.put(`${this.basePath}/modify/${code}`, book, {responseType : 'text'});**

**}**

**}**

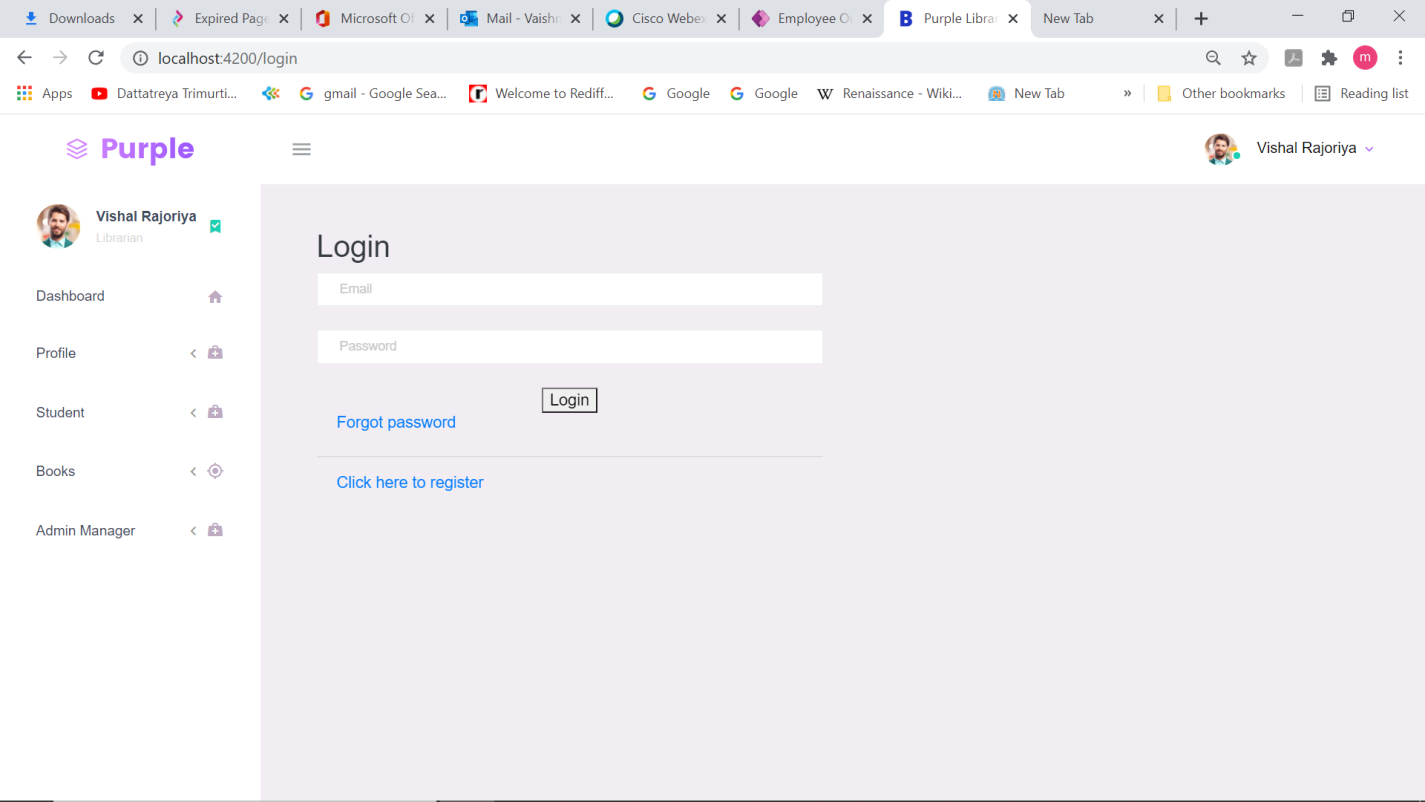
similarly we create all the other service.ts files for the controllers specified in the backend

## Running the Application:

To accomplish this, let's first run the Spring Boot application, so the REST service is alive and listening for requests.

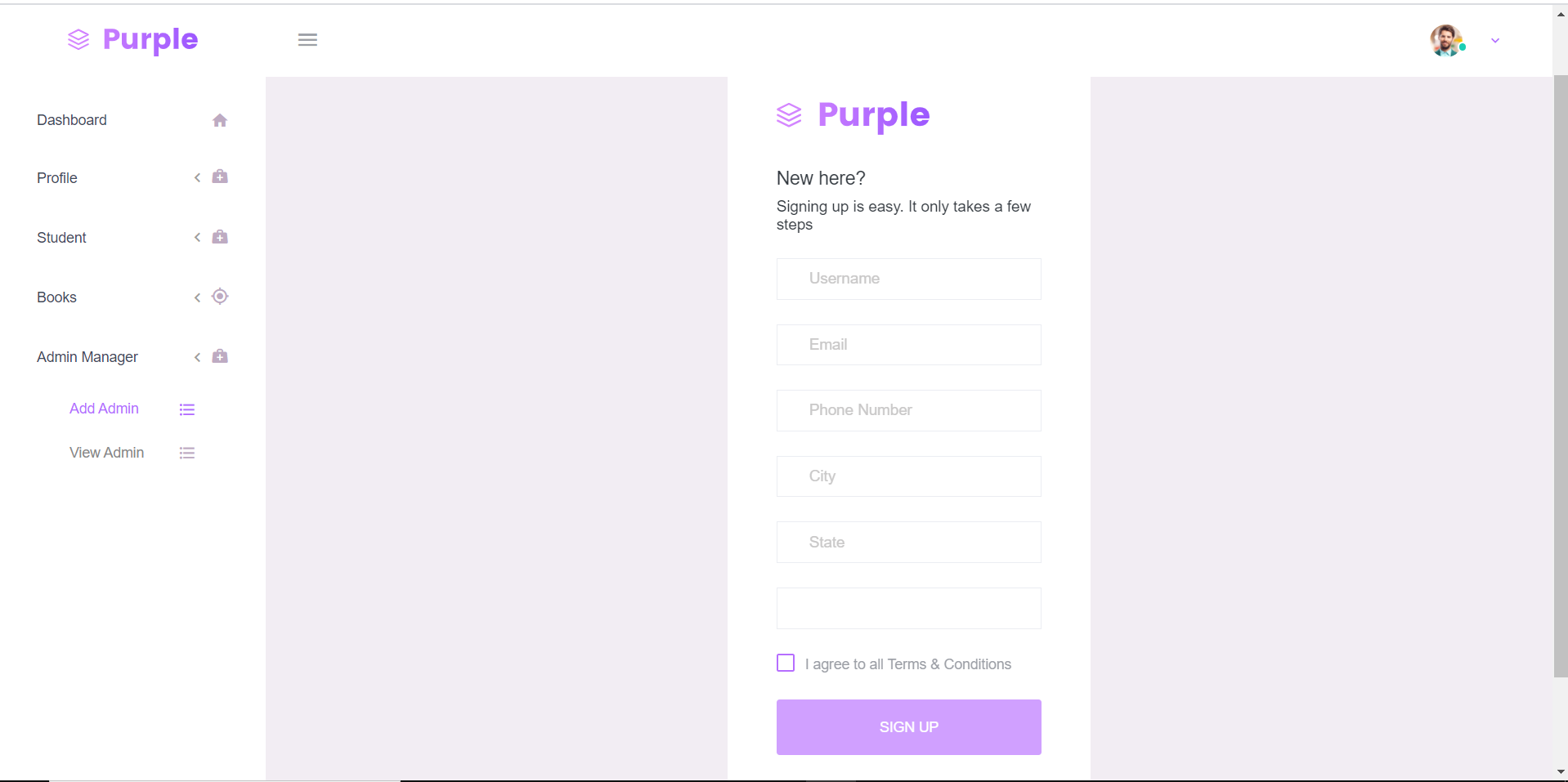
Once the Spring Boot application has been started, let's open a command console and type the following command:

Once the angular application is compiled successfully On giving <http://localhost:4200/> a Login page of our application “ purple-library” appears as shown below:



If we are new users click on register to login.

On click on register a registration page opens:



The Sign up page is provided with validators :

// UserName Validations

  nameFlag: boolean= false;

  validateName() {

    var flag =  /^[a-zA-Z ]+$/.test(this.user['name']);

    if(!flag) {

      this.nameFlag=true;

    }

    else {

      this.nameFlag=false;

    }

  }

  // UserPhone valdiations

  phoneFlag:boolean=false;

    validatePhone(){

        let phone=String(this.user['contactno']);

        if(phone.length!=10){

            this.phoneFlag=true;

        }else{

            this.phoneFlag=false;

        }

    }

    //UserEmail Validation

    emailFlag:boolean=false;

    validateEmail(){

        var flag=/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(this.user['email']);

        if(!flag){

            this.emailFlag=true;

        }else{

            this.emailFlag=false;

        }

    }

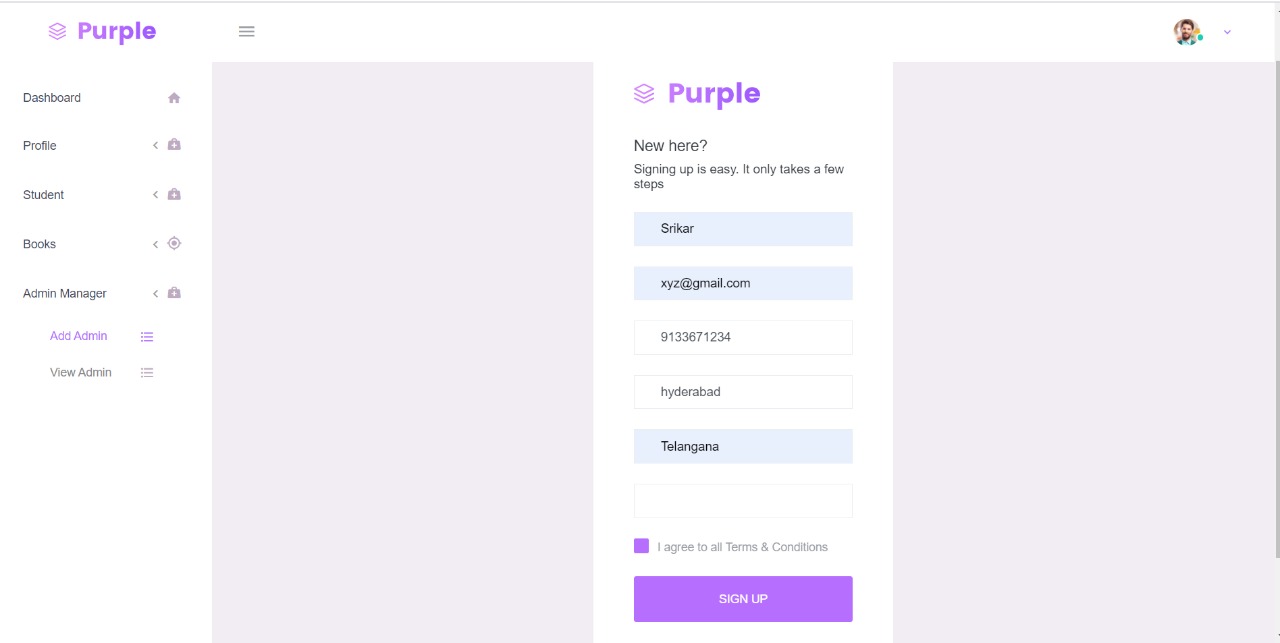
    buttonFlag:boolean=true;

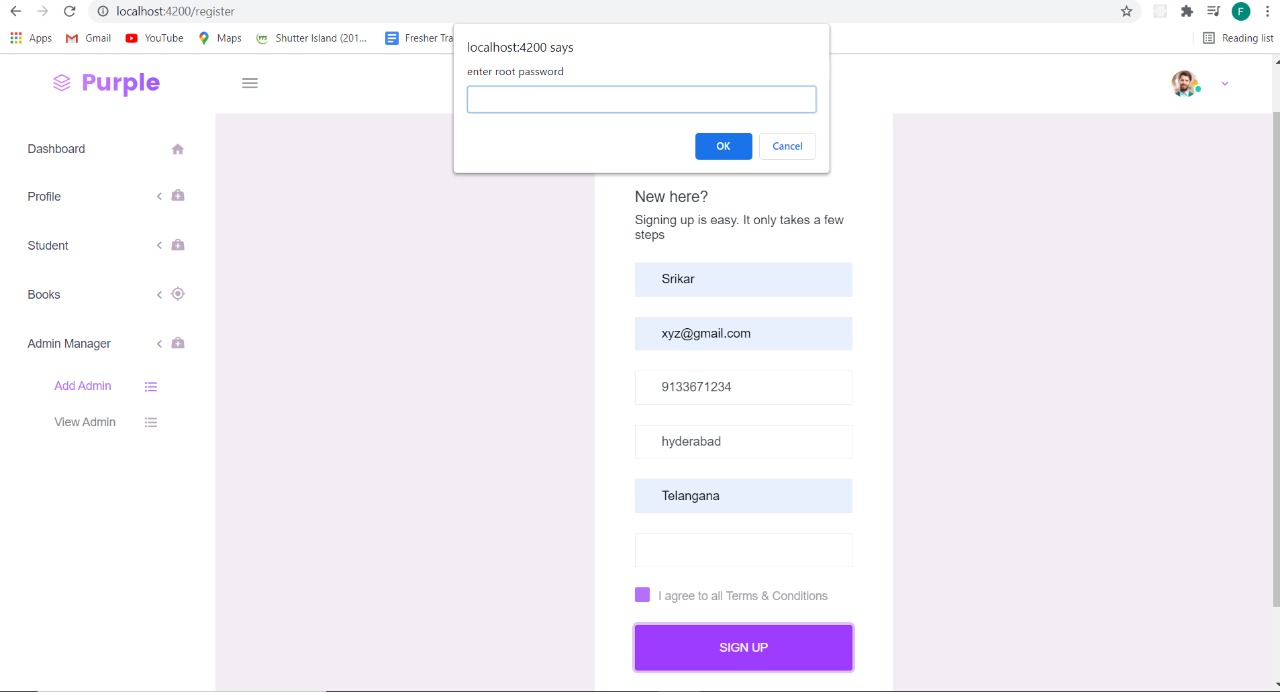
    enableButton(){

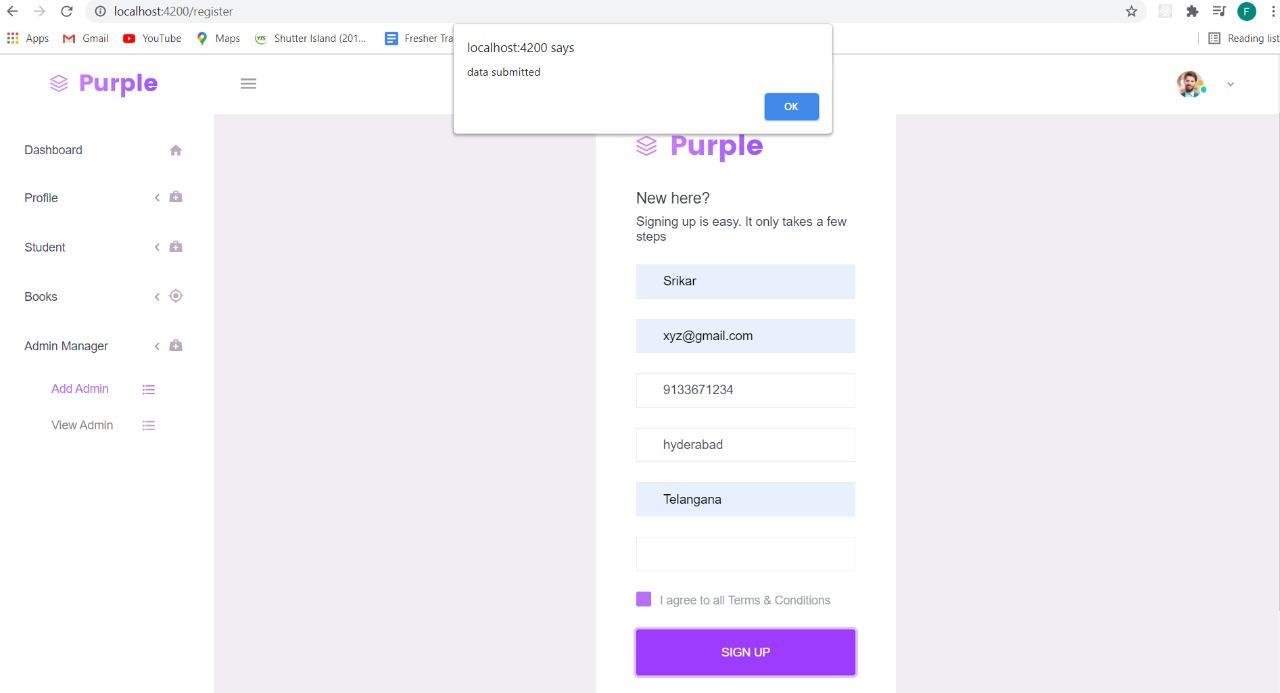
        this.buttonFlag=this.nameFlag||this.emailFlag||this.phoneFlag;

    }

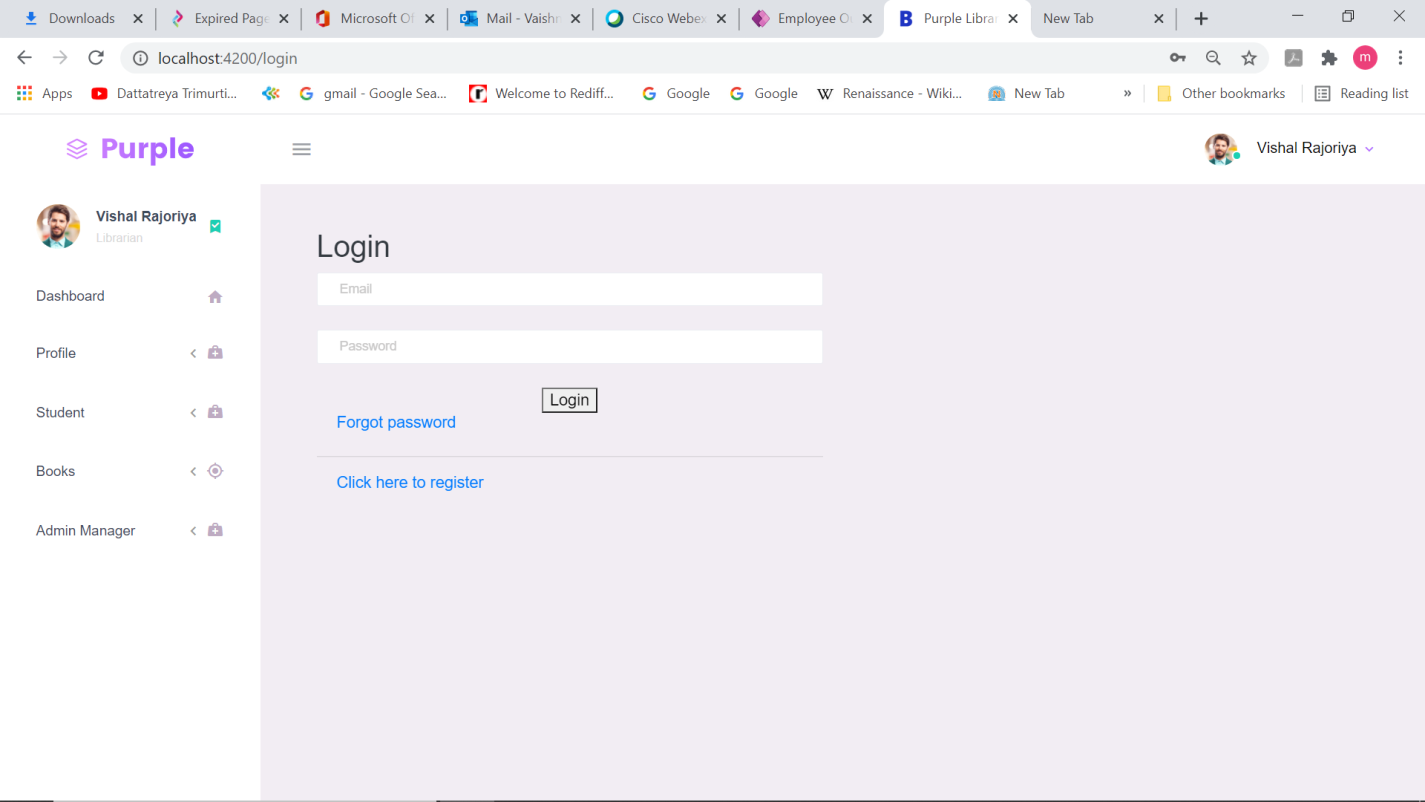
After entering details according to validations then submit button enables and the popup displays where admin has to accept the register.

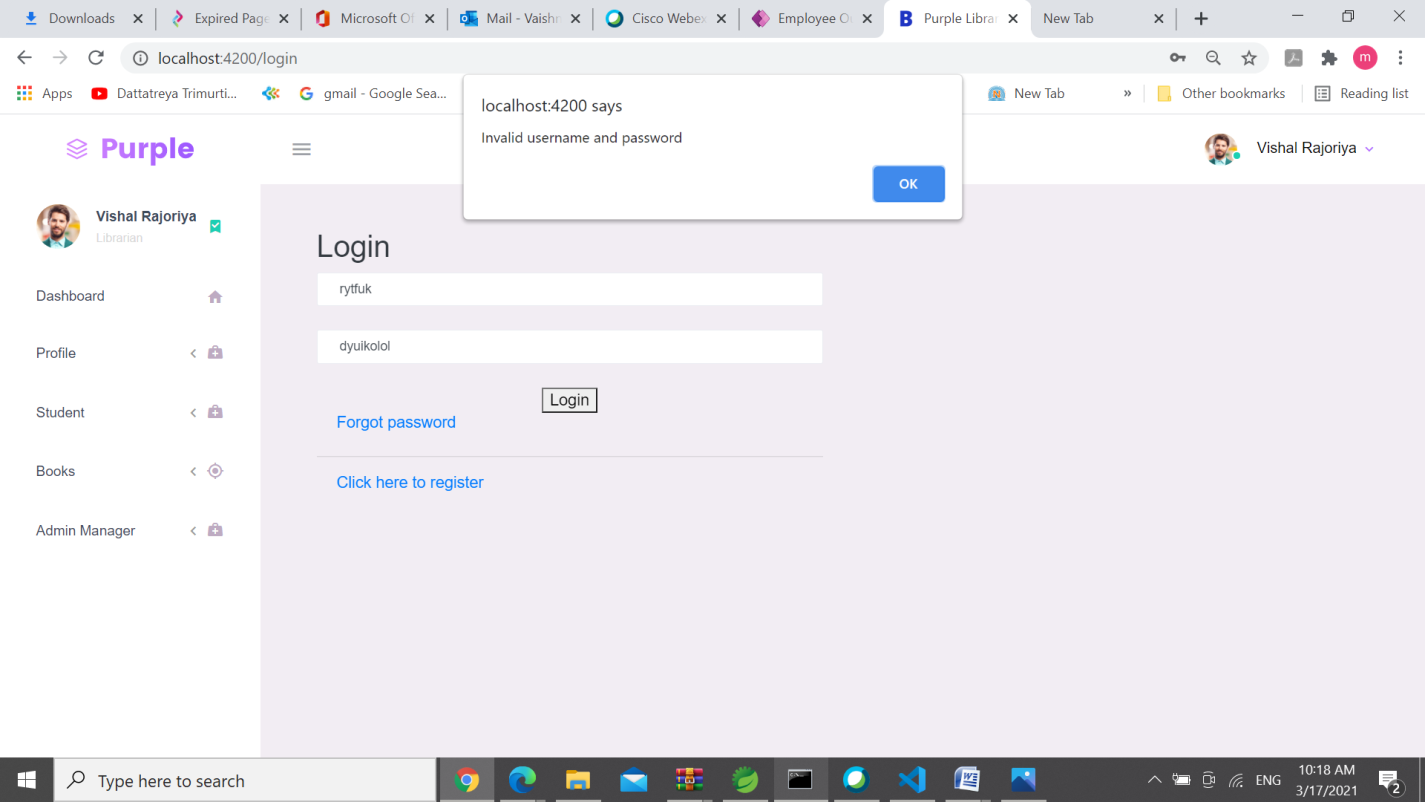




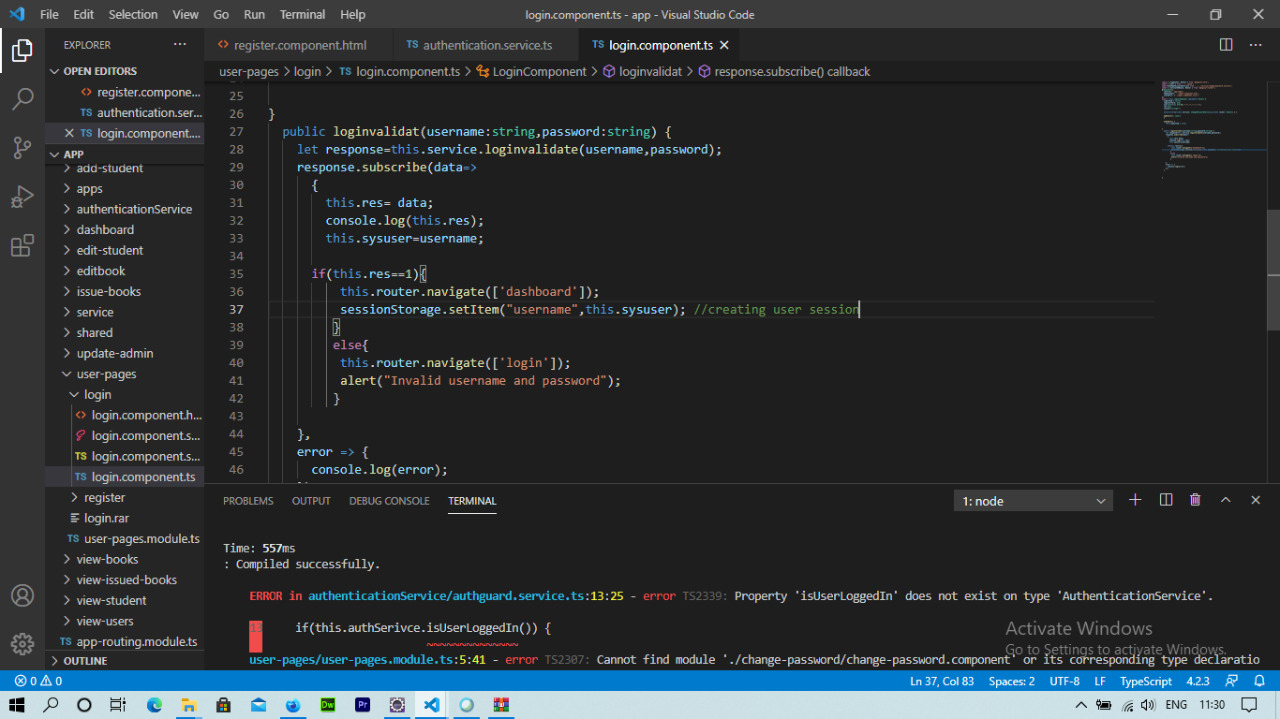


If registration is successful then it redirects to login page.



On entering the details of email and password correctly only it redirects to home page else an alert message occurs.





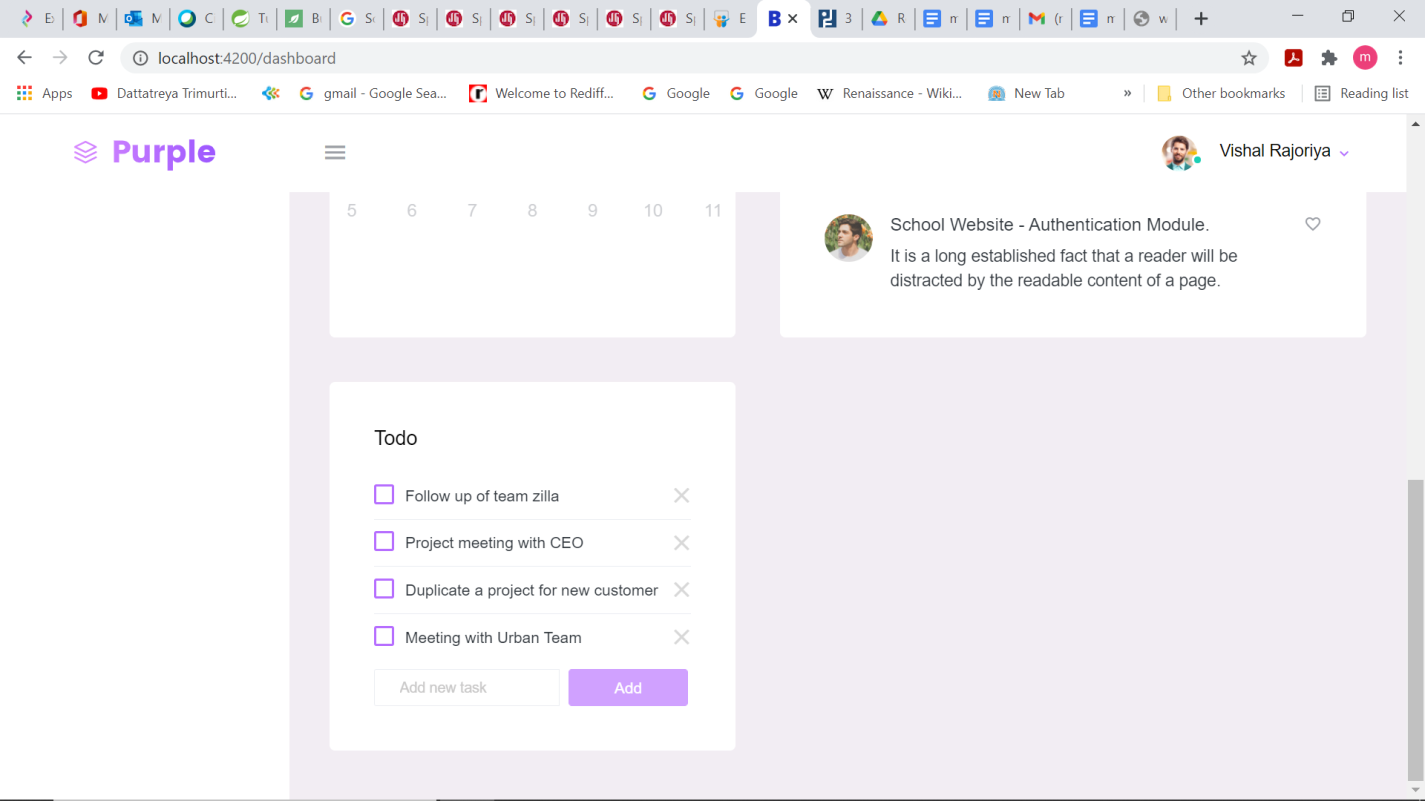
On entering the valid registered email,password only login becomes successful and redirects to dashboard of our project as shown below.

We have created a session so.once we finish login the session gets updated and the name will displays on dashboard.

# 

## From the Above page we can able to see the following Activities performed:

* There is a navbar which provides links for other pages like Profile,Student,Books,Admin .
* By clicking on them we will be redirected to the respective pages.
* The nav bar alsod displays the no of books available,calendar and **to do list component**



In angular to do list component is created

export class TodoComponent implements OnInit {

  form;

  todoArray = [

    { task : 'Meeting with Urban Team' , completed : false },

    { task : 'Duplicate a project for new customer' , completed : false },

    { task : 'Project meeting with CEO' , completed : false },

    { task : 'Follow up of team zilla' , completed : false },

    { task : 'Level up for Antony' , completed : false }

  ];

  constructor(fb: FormBuilder) {

    this.form = fb.group({

      todoitem : ['', Validators.required]

    });

  }

 ngOnInit() {

  }

  addTodo() {

    let newTodoList = { task: '' , completed: false };

    newTodoList.task= this.form.value.todoitem;

    this.todoArray.push(newTodoList);

    this.form.reset();

  }

  removeTodoItem(item) {

   for(let i=0; i<=this.todoArray.length; i++) {

     if(item === this.todoArray[i]) {

       this.todoArray.splice(i, 1);

     }

   }

  }

  changeTodoStatus(event,index) {

    if(event.target.checked) {

    this.todoArray[index]['completed'] = true;

    } else {

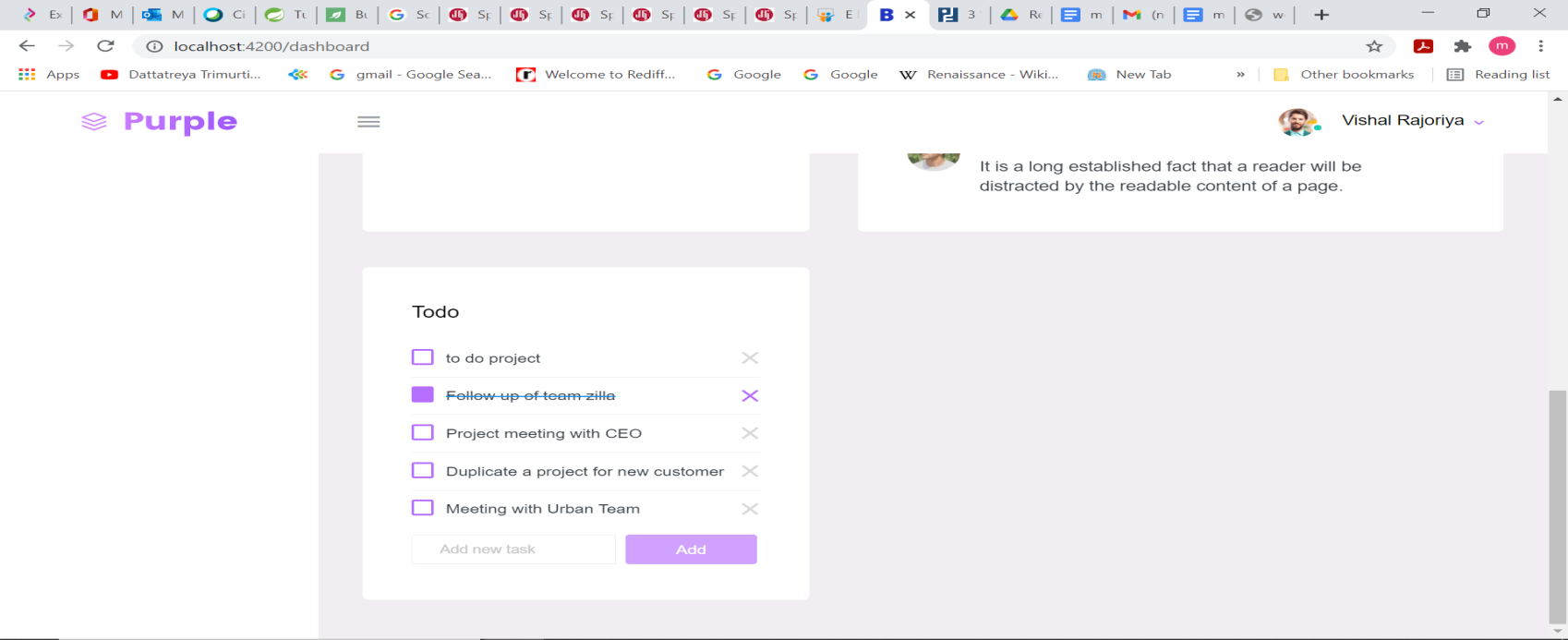
      this.todoArray[index]['completed'] = false;

    }

  }

}

**By clicking on the ADD button we can insert the the list and also we can delete it.**



**If we click on student in dashboard to add student the add student page appears.**

# Add Student page is shown below

# 

**It was all validated :**

form = new FormGroup({

    name: new FormControl('', [Validators.required, Validators.minLength(3)]),

    address: new FormControl('', [Validators.required, Validators.minLength(3)]),

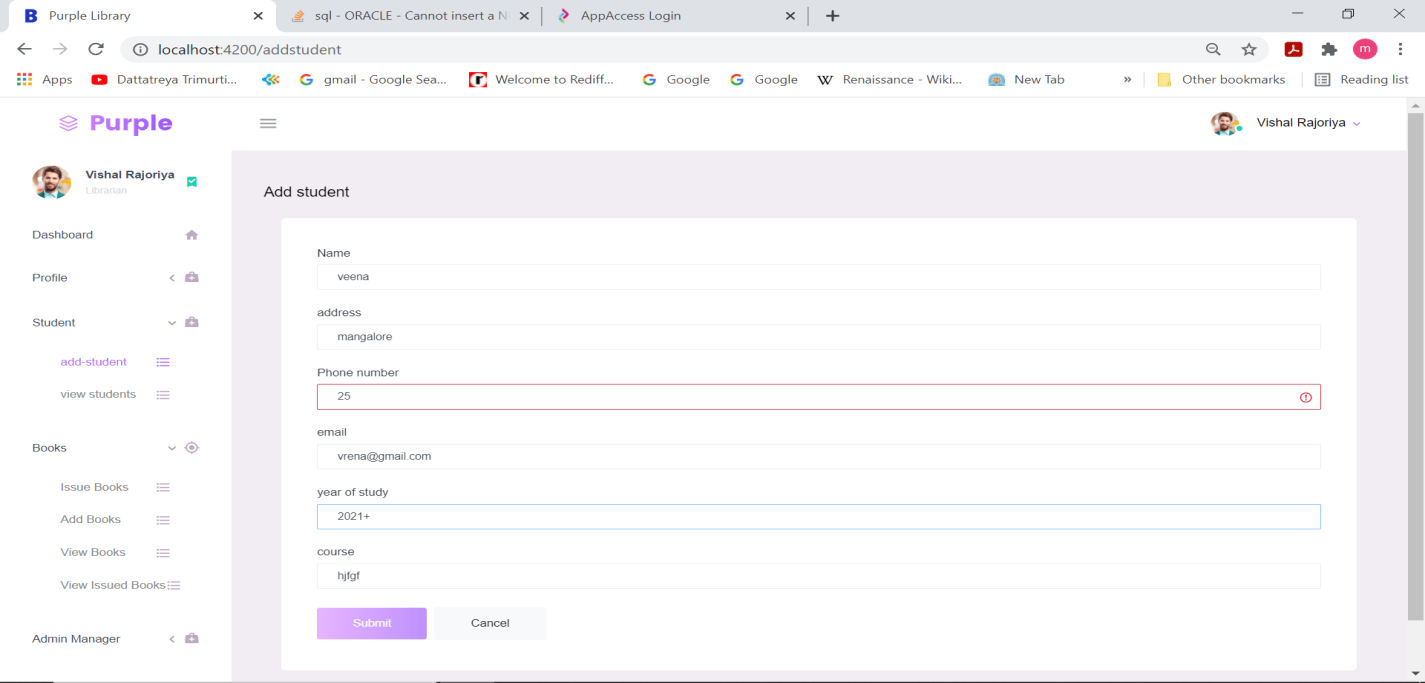
    phone: new FormControl('', [Validators.required, Validators.minLength(10)]),

    email: new FormControl('', [Validators.required, Validators.minLength(6)]),

    year: new FormControl('', [Validators.required]),

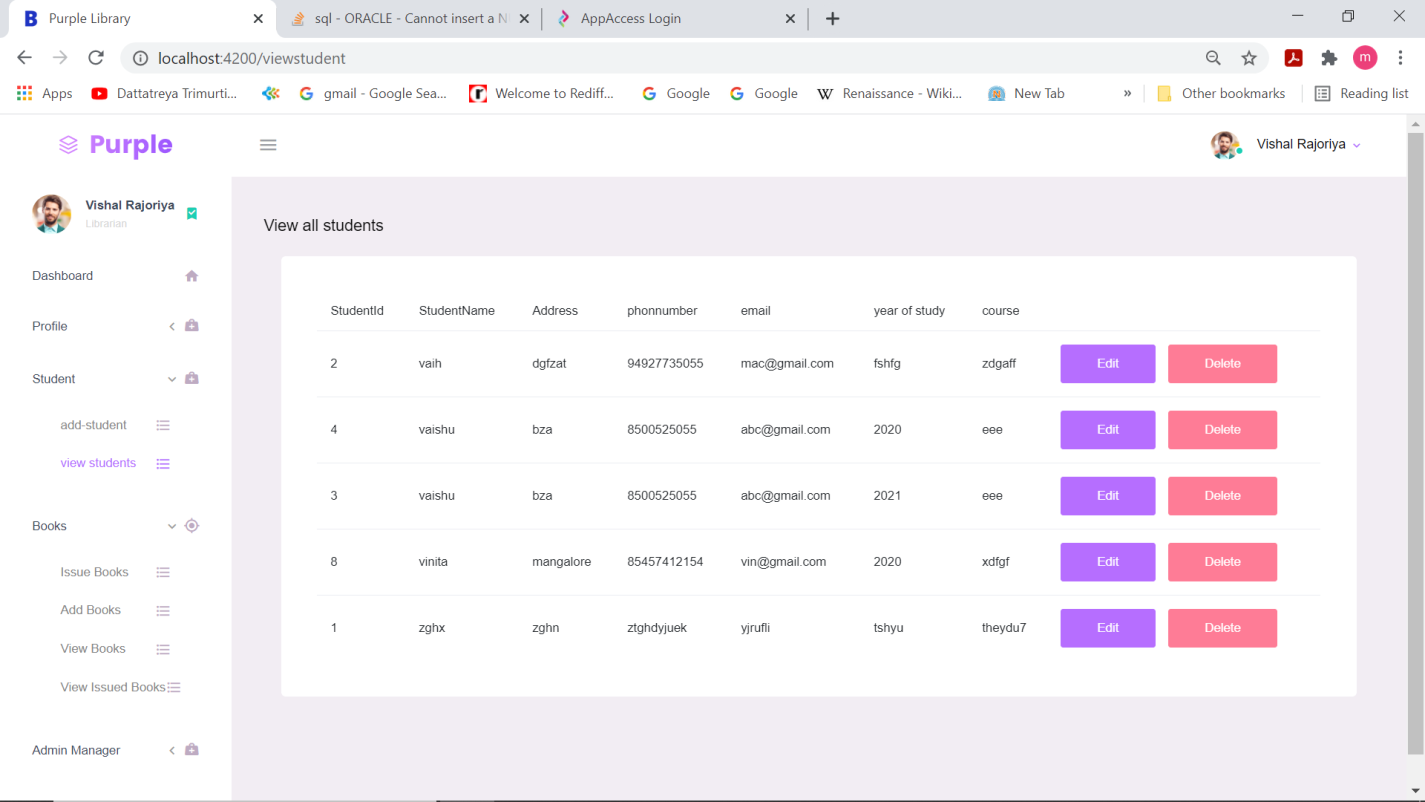
    course: new FormControl('', [Validators.required, Validators.minLength(3)])

**If any of validation is missing it will not get submit.**

****

**Now to view the students added click on view students to view.**

View Student page is shown below

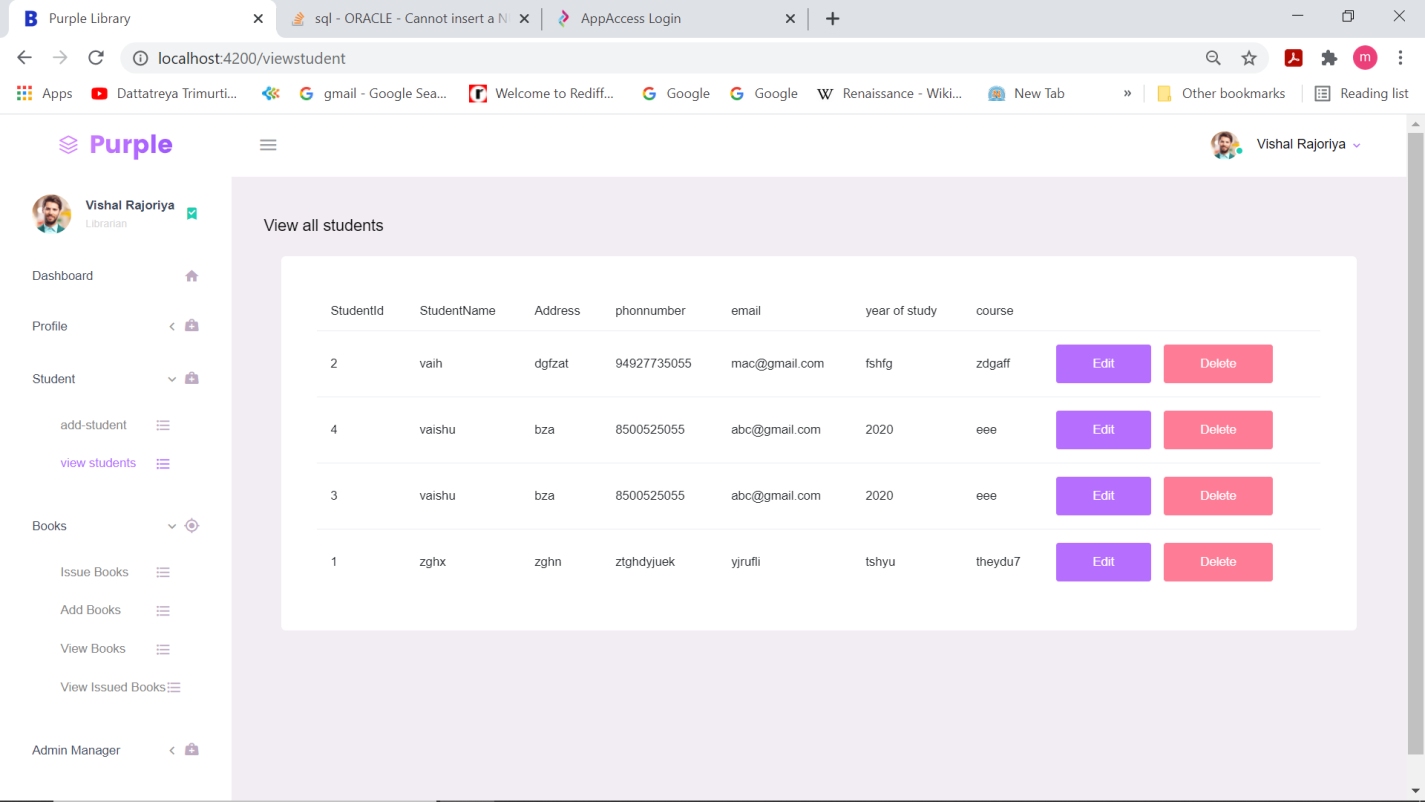
****

**TO EDIT ANY DETAILS OF THE STUDENT OPTION IS GIVEN TO EDIT AND DELETE.**

**IN THE ABOVE TABLE STUDENT ID 3 YEAR OF STUDY IS EDITED TO 2020 AND THE TABLE IS DISPLAYED BELOW.**

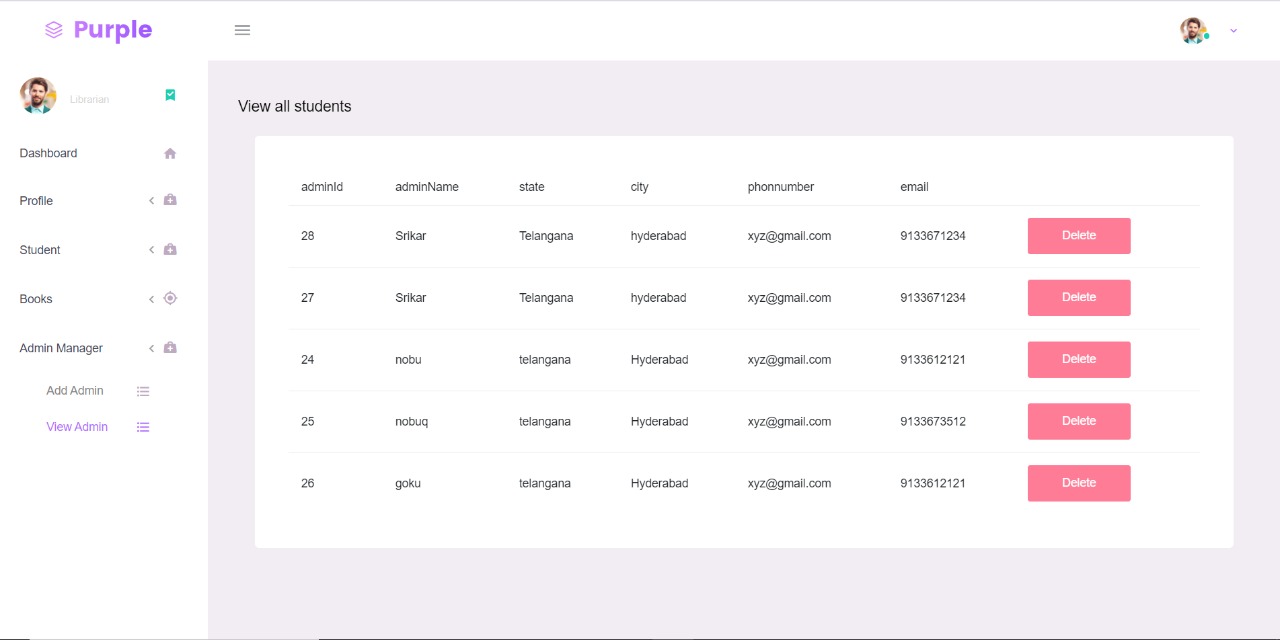
**AND THE STUDENT ID WITH 8 IS DELETED.**

**THE UPDATED TABLE IS:**

****

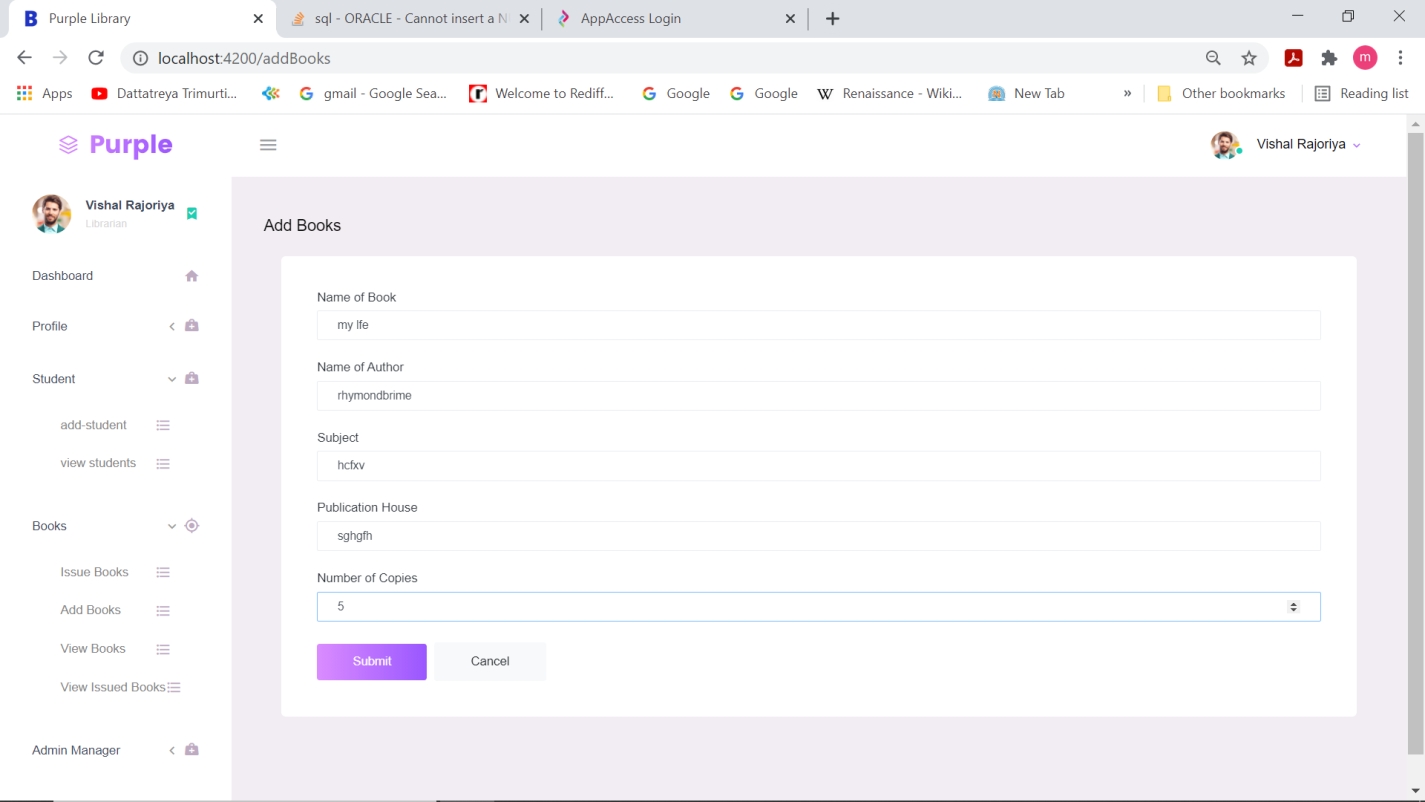
**In dashboard if we click on admin then all the registered ones who are been added as admin are visible.**

VIEW ADMIN PAGE

****

**In dashboard if we click on add books:**

ADD BOOKS PAGE

****

Validators provided:

 book\_name: new FormControl('', [Validators.required, Validators.minLength(3)]),

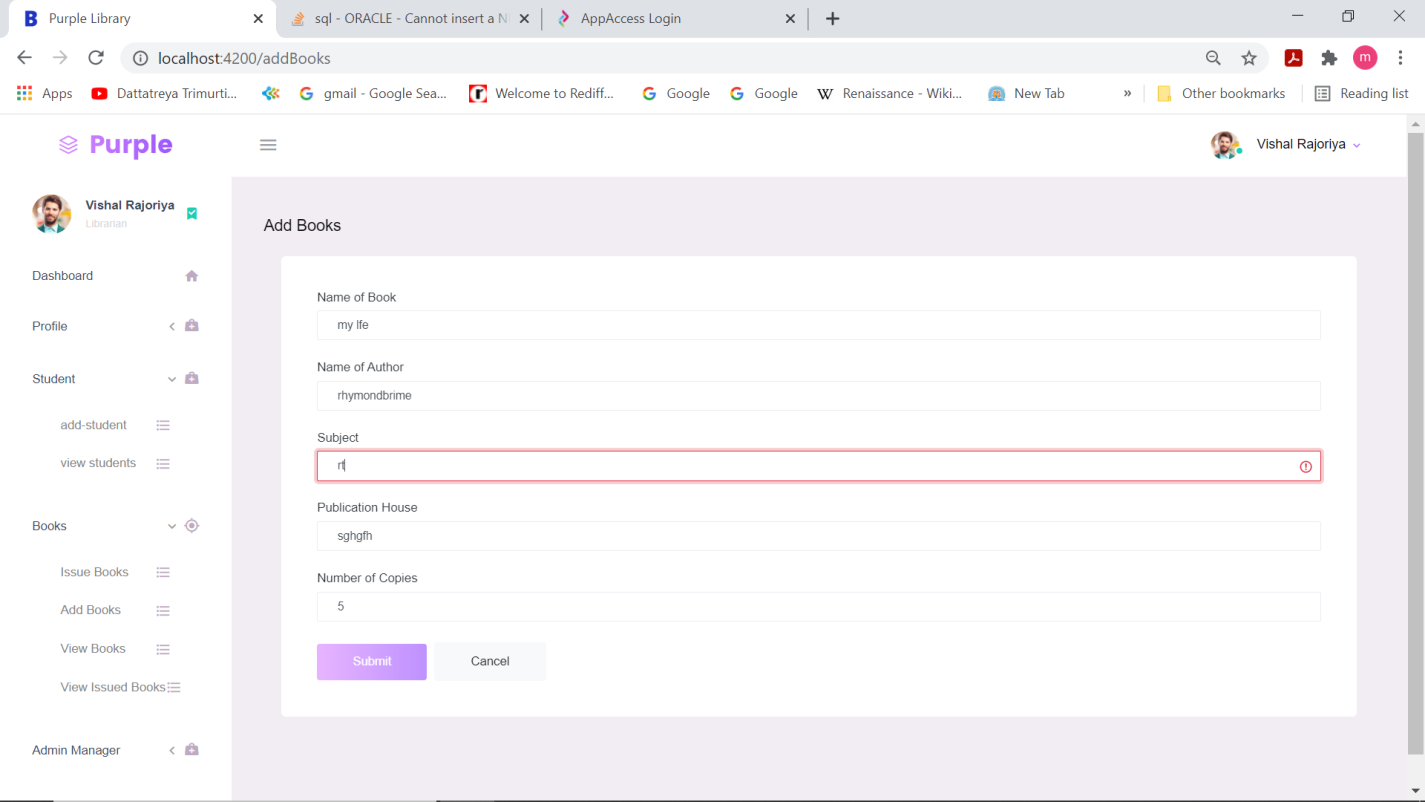
    author: new FormControl('', [Validators.required, Validators.minLength(3)]),

    subject: new FormControl('', [Validators.required, Validators.minLength(3)]),

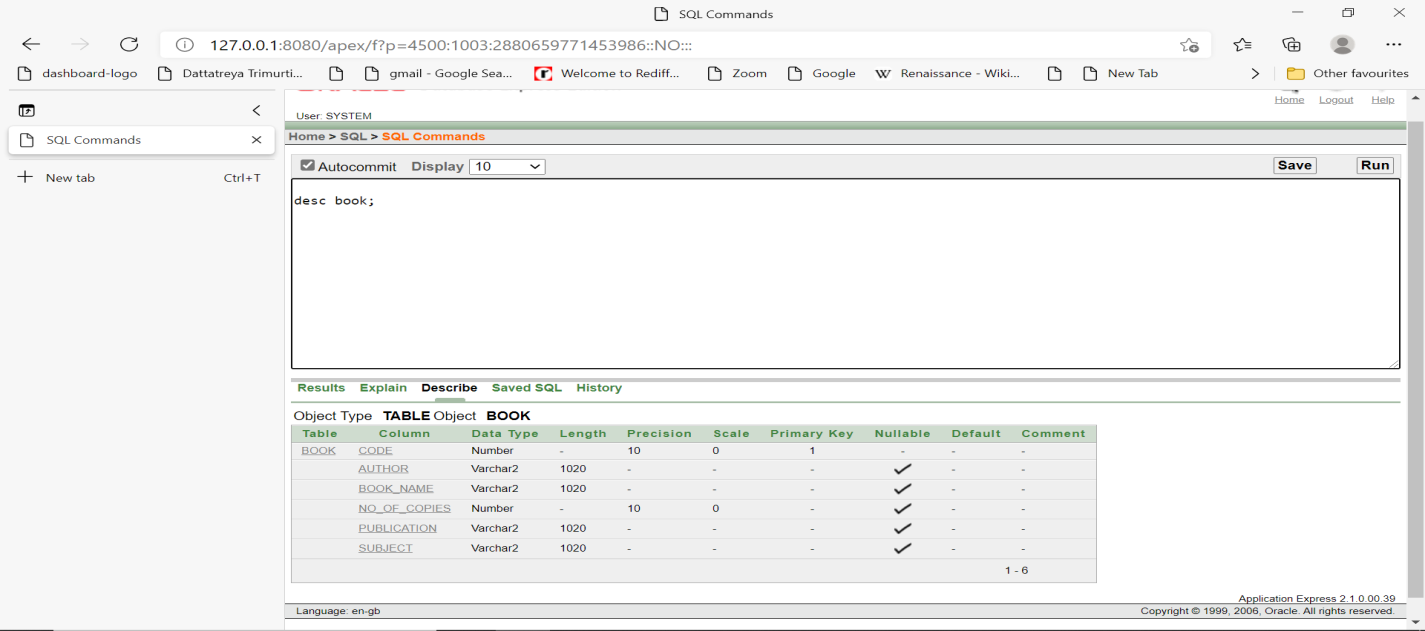
    publication: new FormControl('', [Validators.required, Validators.minLength(3)]),

    no\_of\_copies: new FormControl('', [Validators.required, Validators.minLength(3)])

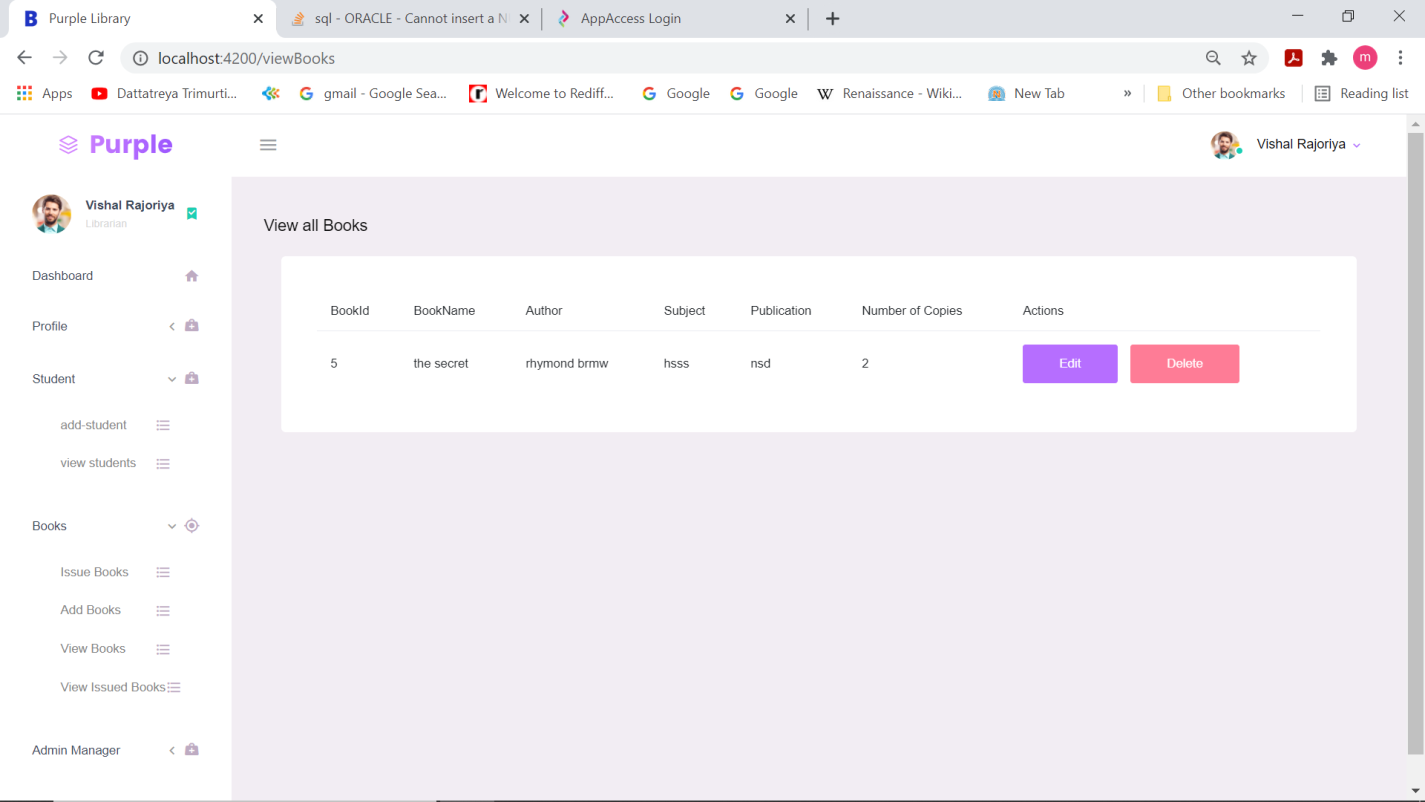
If any validators are not satisfied then the submit button will never be enabled.



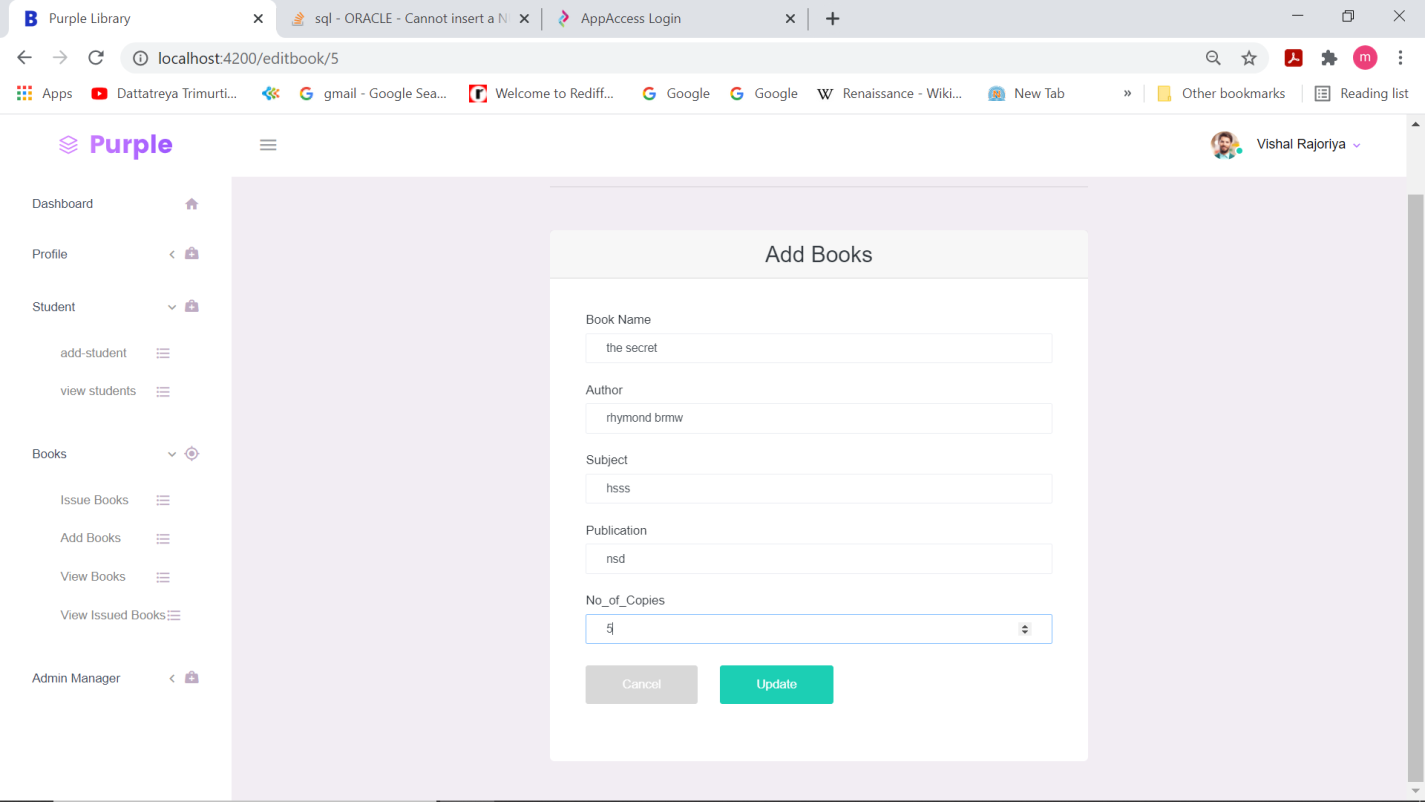
Fetched from the database from book table created automatically by hibernate.



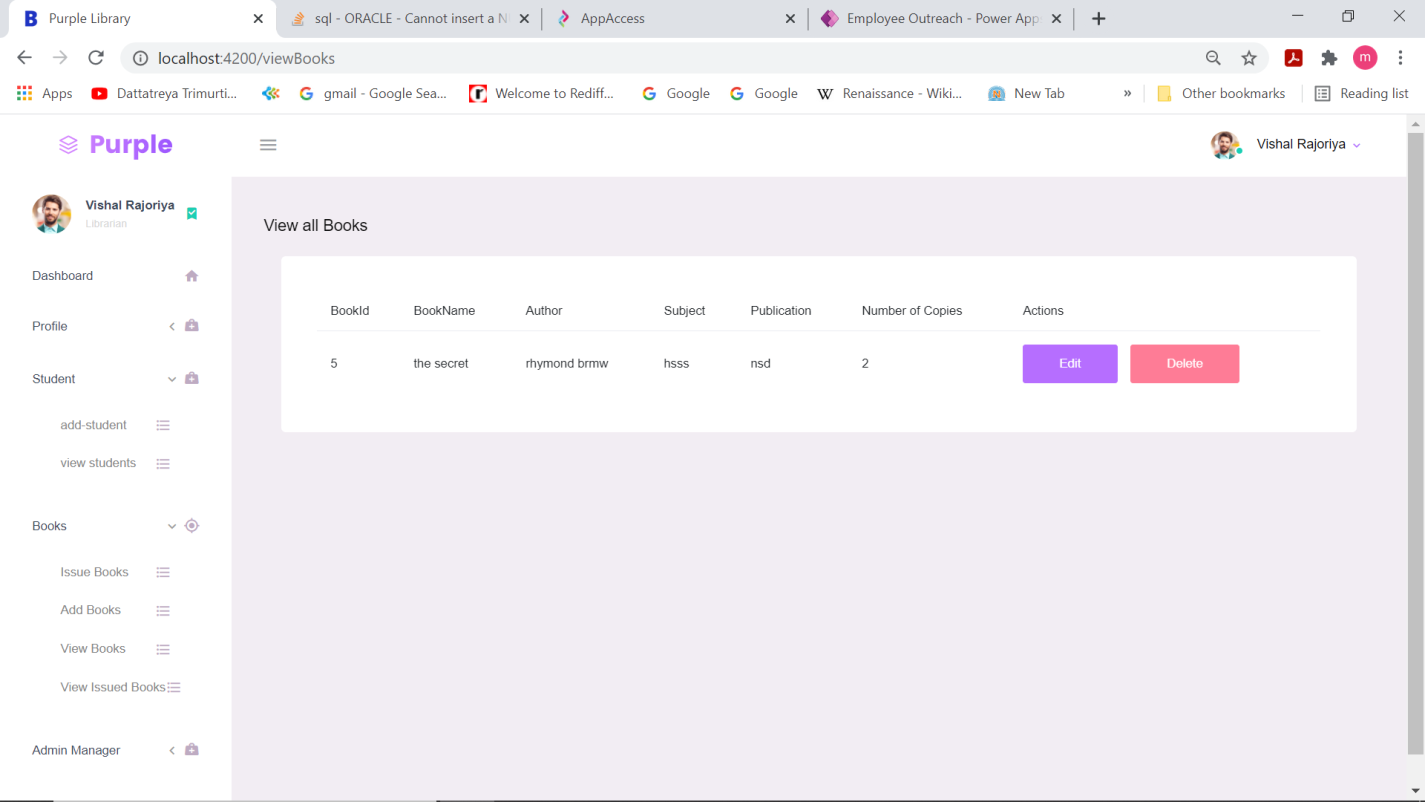
Now after submitting to view the books added click on view books to view the books.



Now we can also edit the book operations by edit and delete buttons provided.

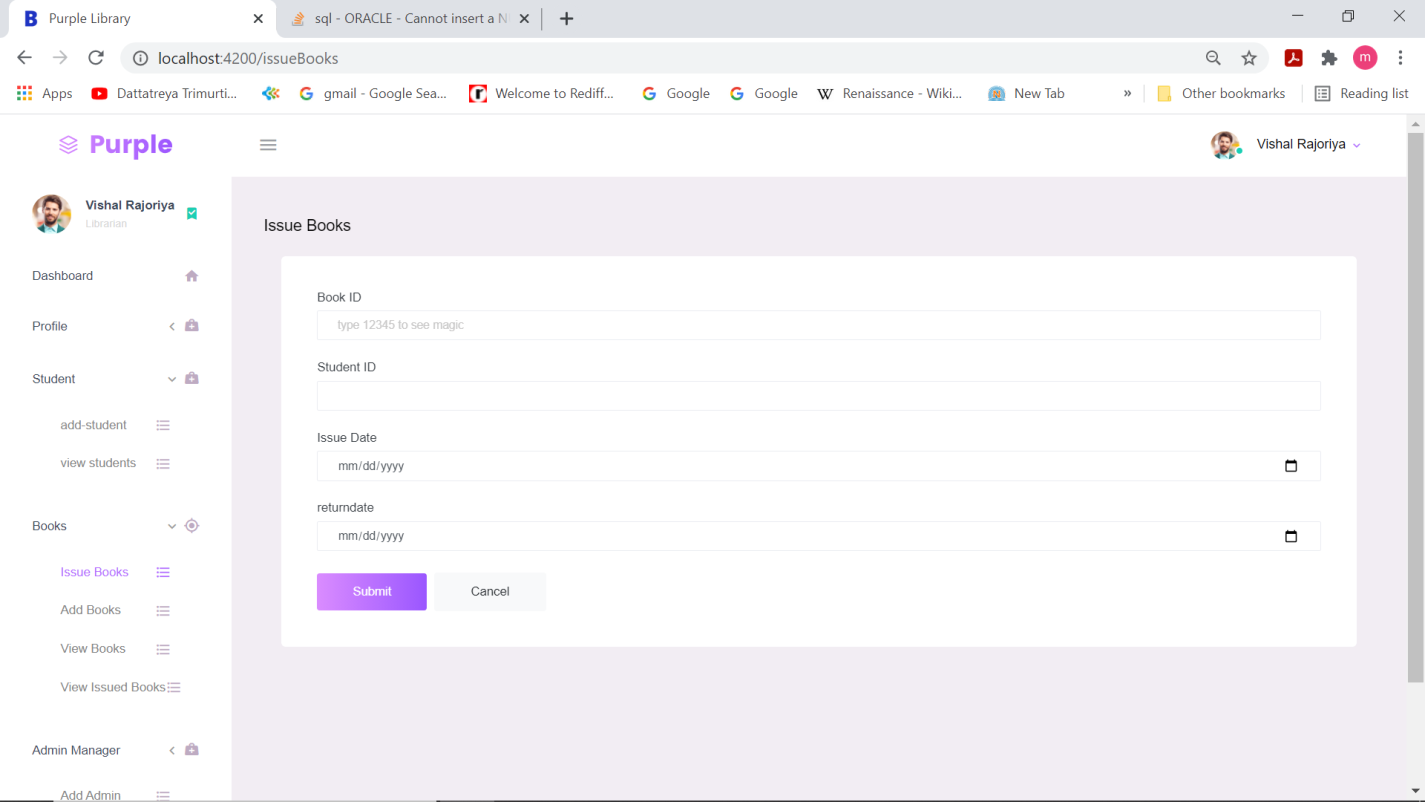


Now I have edited no.of copies and after updating the table is as follows.



Now to issue books click on issue books.

**ISSUE BOOKS PAGE**



**If we give any book id which is there in the view books only it will accept.**

**ANGULAR CODE FOR THIS IS:**

**@Component({**

**selector: 'app-issue-books',**

**templateUrl: './issue-books.component.html',**

**styleUrls: ['./issue-books.component.scss']**

**})**

**export class IssueBooksComponent implements OnInit {**

**form: FormGroup;**

**submitted = false;**

**showBook:boolean=true;**

**showstu:boolean=true;**

**student:Student;**

**book:Book;**

**issuebook:Issuebook;**

**// showBook:boolean=this.validationservice.showBook;**

**constructor(private formBuilder: FormBuilder,private validationservice:ValidationService,private issuebookservice:NgserviceService) { }**

**ngOnInit(){**

**this.form = this.formBuilder.group({**

**book\_id:['', [Validators.required], this.validationservice.validateBookId.bind(this.validationservice)],**

**student\_id:['', [Validators.required], this.validationservice.validateStuId.bind(this.validationservice)],**

**issuedate:['', [Validators.required]],**

**returndate:['', [Validators.required]]**

**});**

**}**

**get formcontrols() {**

**return this.form.controls;**

**}**

**onSubmit() {**

**this.submitted = true;**

**this.issuebook=new Issuebook(this.form.controls.book\_id.value,this.form.controls.student\_id.value,**

**this.form.controls.issuedate.value,this.form.controls.returndate.value)**

**console.log(this.issuebook);**

**if (this.form.valid) {**

**this.issuebookservice.issueBook(this.issuebook).subscribe(data=>{**

**console.log("data submited");**

**})**

**alert('Form Submitted succesfully!!!\n Check the values in browser console.');**

**}**

**}**

**onupdate(event:any){**

**setTimeout(() => {**

**this.showBook=this.validationservice.showbook;**

**this.book=this.validationservice.book;**

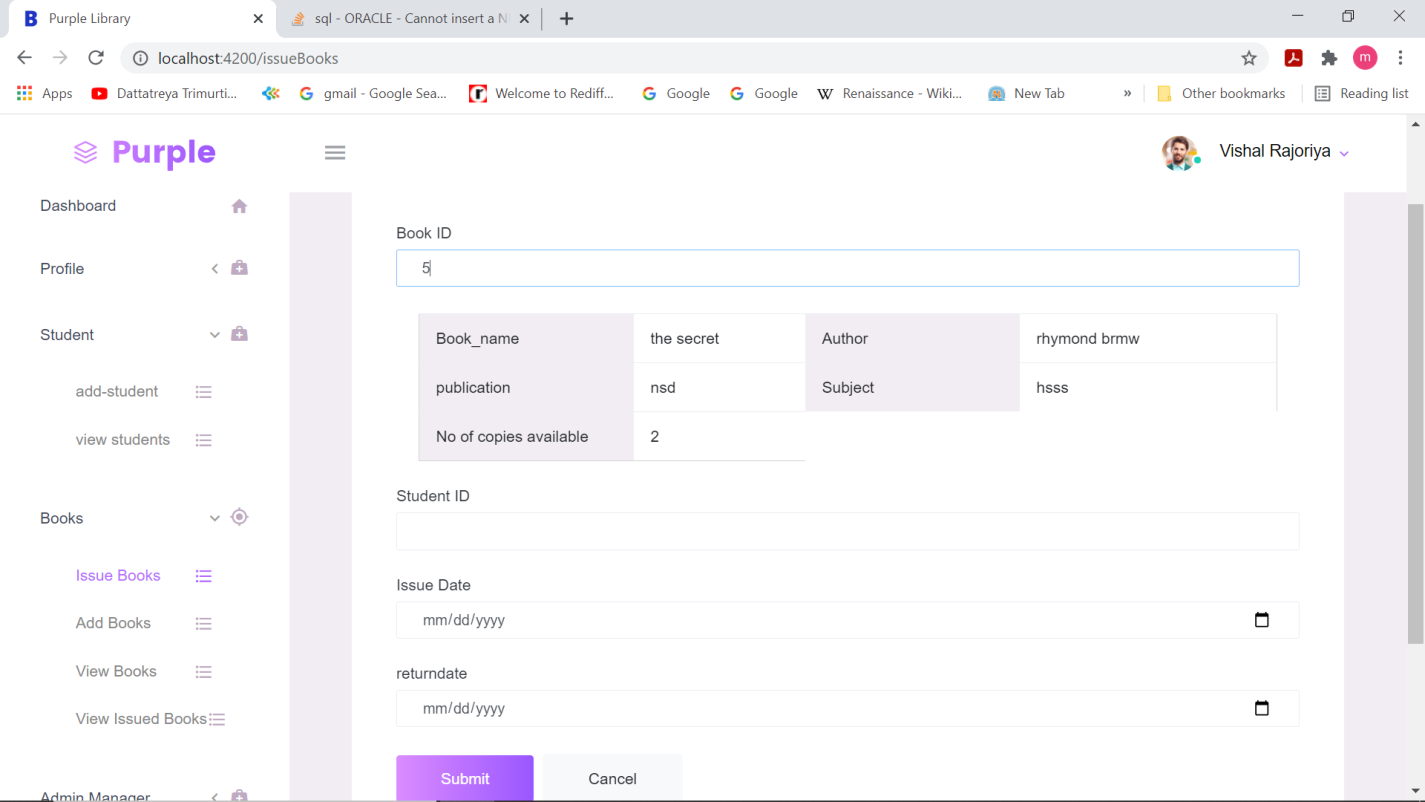
**console.log(this.book);**

**},1000);**

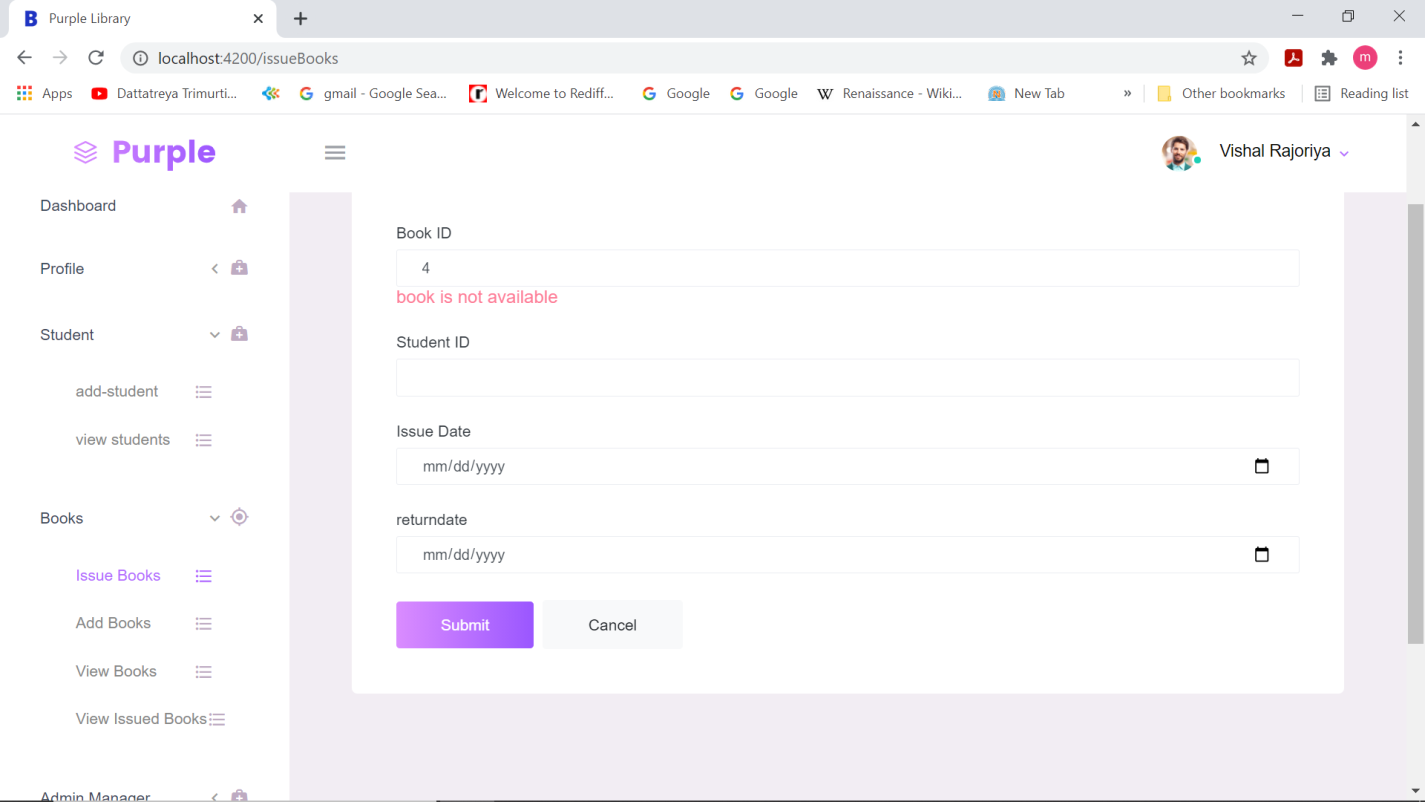
**}**

**}**

**JUST on giving id=5; the details of the book is displayed as shown.**

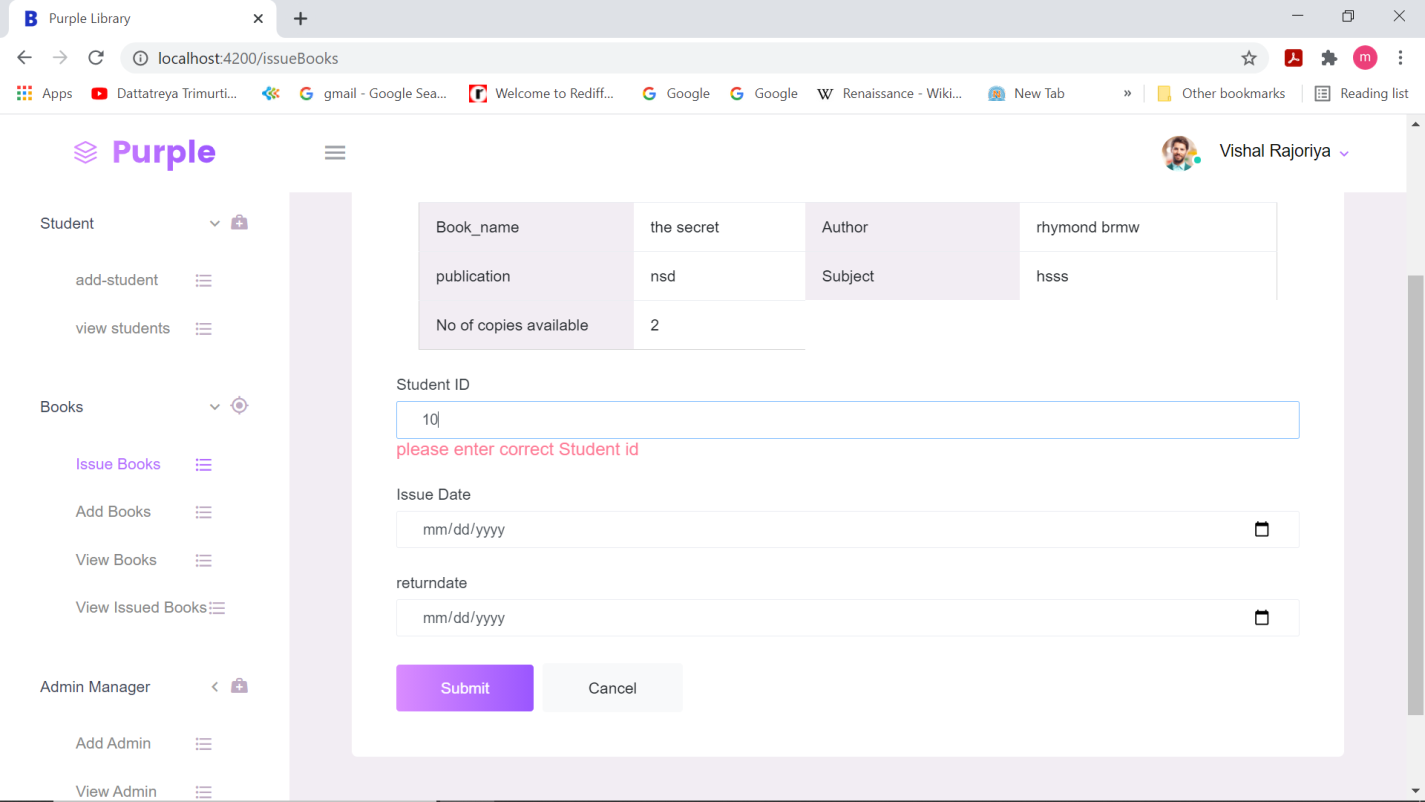
****

**If we give any book id not appearing on view books then “ BOOK IS NOT AVAILABLE” is displayed.**

****

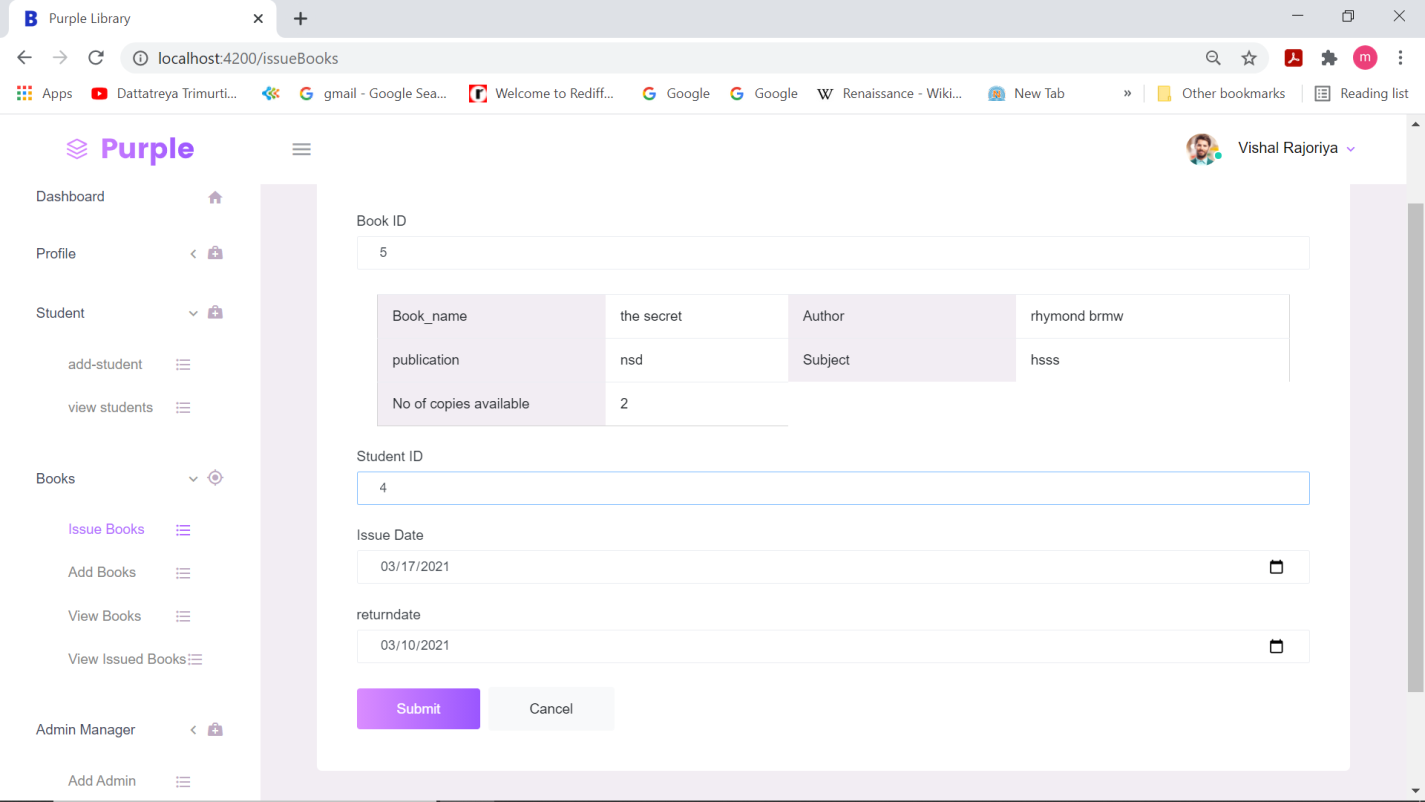
**Now if the student is also not registered then he won’t have access to take books.**

**If any of the student not registered comes to take books**

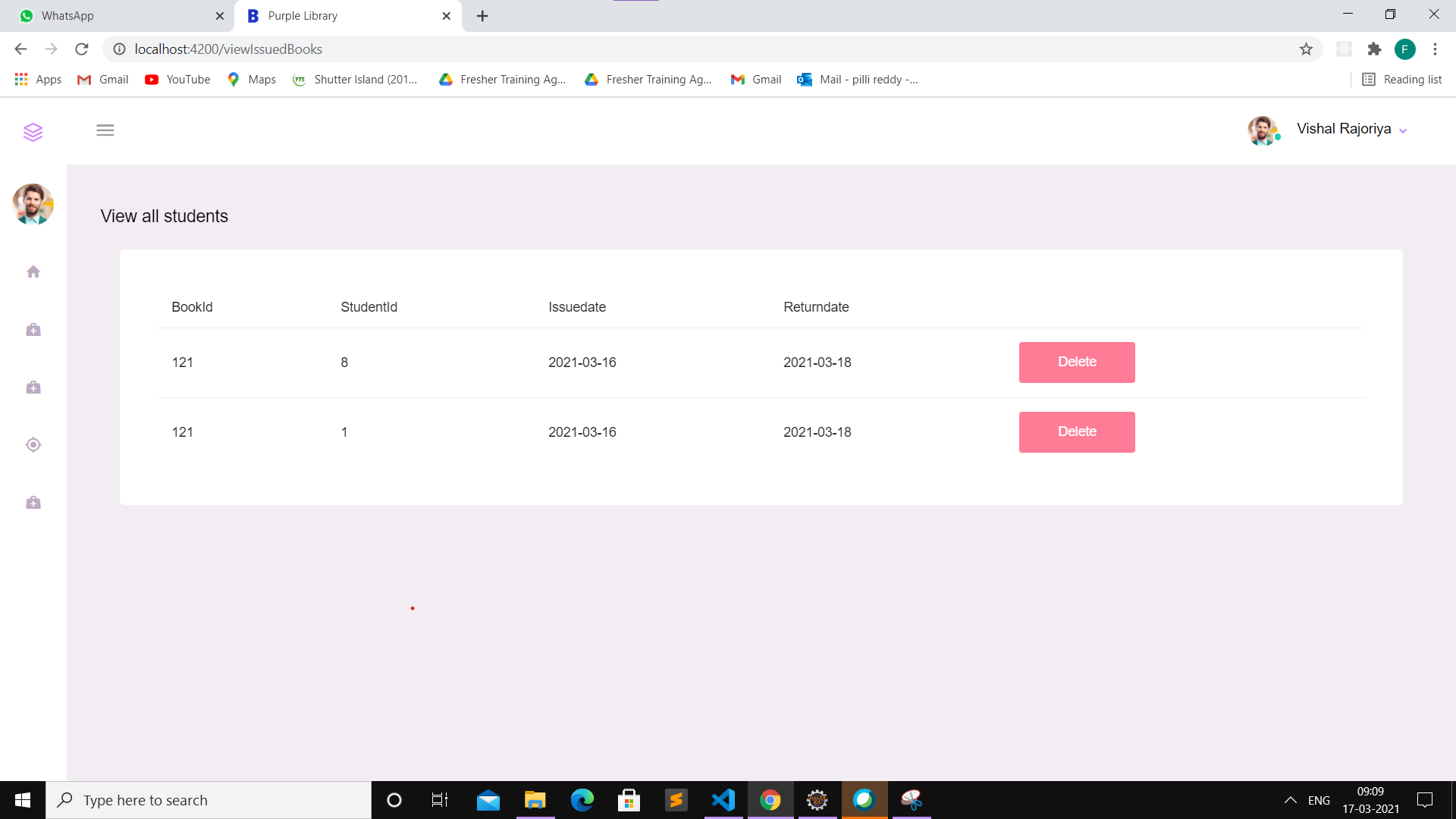
****

**Then this popup appears.**

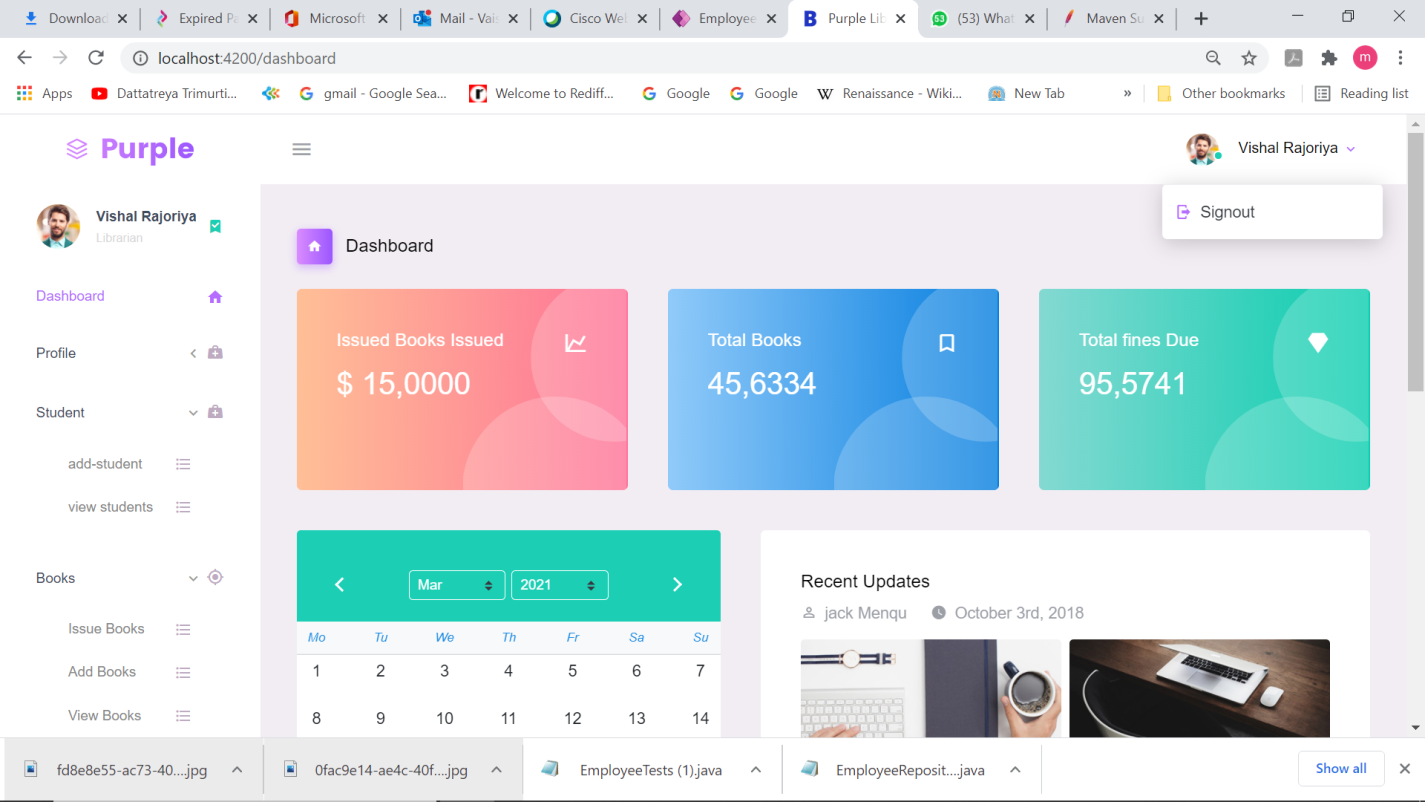
**If all the details are given correctly then only the submit button enables and get submitted.**

****

**Then to see the books issued click on view issue books to view .**

****

**After finishing all the work by clicking on signout in the dashboard the session gets closed .**

****