**DEVELOPMENT PHASE III REQUIREMENTS REPORT**

For

COURSE REGISTRATION SYSTEM

Version 1.0

Prepared by: Team Innovators (Sreekanth Vobilishetty, Akhila Yarlagadda, Piyusha Varshini Tirukovalluru, Meghana Reddy Akkati)

University of North Texas

11/18/2019

**1. ADMIN MODULE FEATURES**

In this module, we plan to implement the admin features which include Updating the courses in Course Edit tab such as course name, instructor, course timings, course cost, course Maximum strength etc. Admin should be able to edit any of the above-mentioned features of the course and when succeeded an email should be generated to intimate the change to all the users. Admin should also be able to delete or add the course to the list of existing courses.

In Finances tab, admin should be able to update changes into the database when a user makes an offline payment.

In Edit grades tab, admin should also be able to update the grades of the students for respective courses to which they’ve enrolled for the following semesters. Email notification should be sent whenever there is an update in the grades for the students.

In Enrollment details tab, Admin should be able to view all the details of the enrollments such as student name, ID and details of course in which they have enrolled when given inputs such as department name, course-id, semester and year.

* 1. **Course Edit**
     1. **Edit**

In this tab, admin can make changes to the courses such as course name, instructor, course timings, course cost, course Maximum strength etc.

* + 1. **Add**

In this tab, admin can add a new course to the existing list of courses.

* + 1. **Delete**

In this tab, admin can delete a course from the list of courses.

* 1. **Finances tab**

In this tab, admin can update the fee details of the students who have made an offline payment.

* 1. **Edit Grades tab**

In this tab, admin can edit the grades of the students of courses which belong to previous semesters rather than current.

**1.4 Enrollment details tab**

In this tab, admin should be able to view all the details of the enrollments such as student name, ID and details of course in which they have enrolled when given inputs such as department name, course-id, semester and year.

**1.5 Development phase III:**

In this phase we have developed the admin module which can perform the following features in respective tabs such as

Courses Edit

Add a course

Delete a course

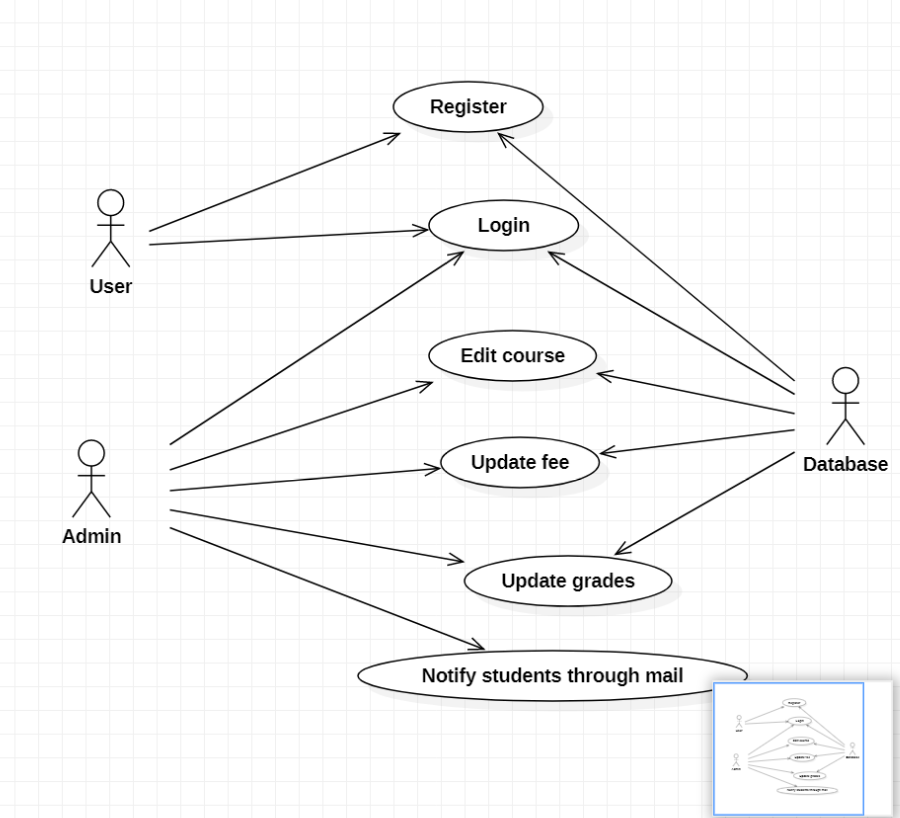
Finance tab

Edit grades tab

Enrollment details tab

**2. UML DIAGRAMS**

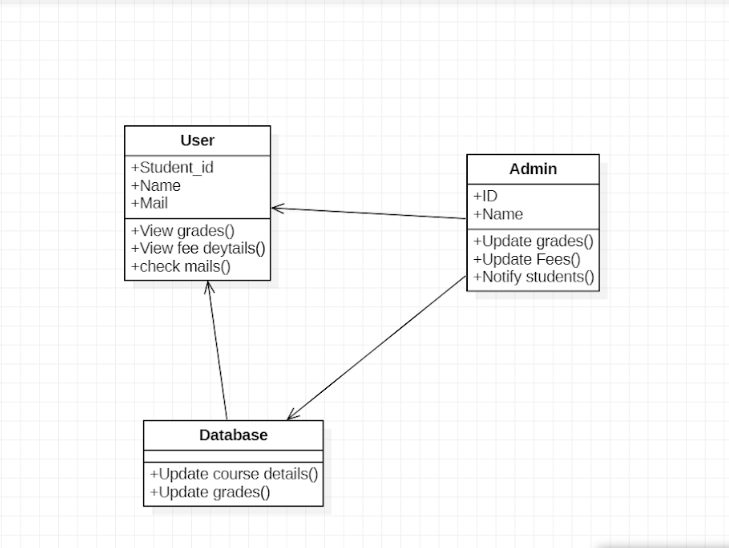
**2.1 Use case Diagram**

****

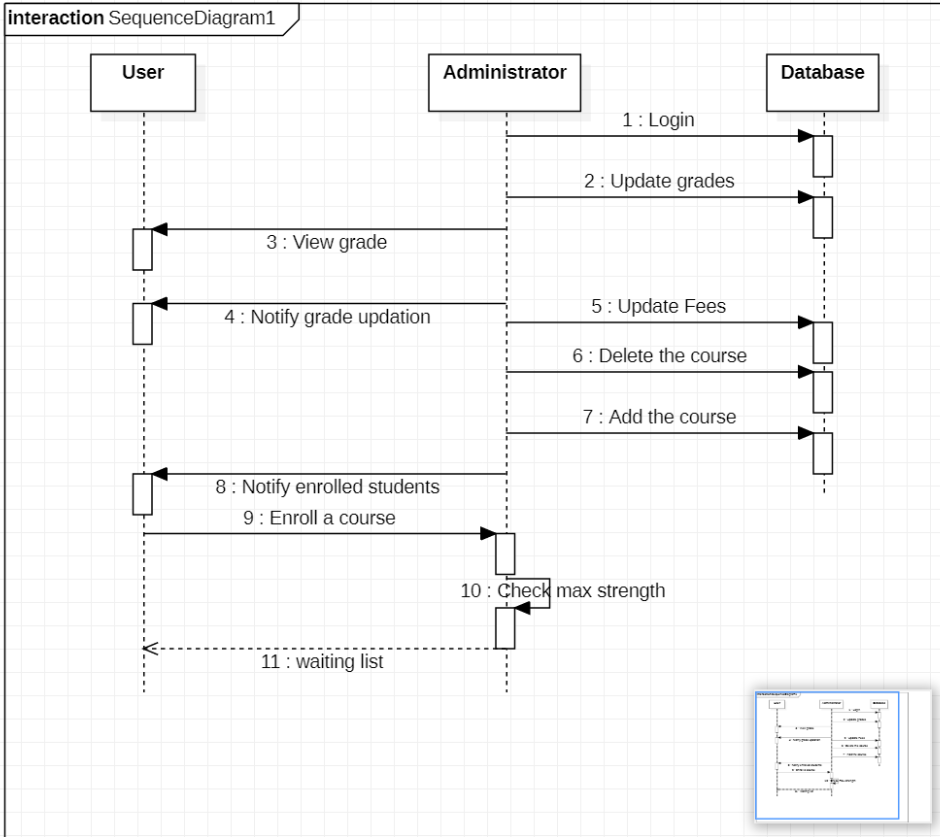
The above Use case diagram indicates three actors namely User, Admin and Database where some of them are directly associated with the use cases mentioned above such as Register, Login, Edit course, Update fee, Update grades, email notification.

The User has no direct association with all the use cases mentioned in the diagram. The admin has a direct association for all of the use cases mentioned whereas database is directly associated with all the use cases since all the data needs to be stored and accessed from the database itself.

**2.2 Class Diagram**

****

**2.3 Sequence Diagram**

****

The above diagram displays the sequence of actions which takes place in an order from the time they occur among User, database and admin.

**3.TESTCASES**

**3.1 UNIT TESTING WITH SPRING BOOT AND JUNIT**

**3.1.1 Unit Testing**

We want to create a Unit test for the Controller class which is Registration Controller in our project which has several Get and Post methods which needs to be tested.

In the Unit testing,

We will mock out the Registration Service using Mockito.

We will use Mock MVC framework to launch only the Registration Controller.

**3.1.2 JUnit Test code**

package com.unt.registration.controller;

//package com.javainuse.test;

import static org.junit.Assert.\*;

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

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

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

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

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

import org.junit.Test;

import org.mockito.Mockito;

import org.skyscreamer.jsonassert.JSONAssert;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.RequestBuilder;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.unt.registration.service.\*;

@RestController

public class RegistrationController {

@RunWith(SpringRunner.class)

@WebFluxTest(controllers = RegistrationController.class)

public class RegistrationControllerTest {

@Autowired

private RegistrationService registrationService;

@GetMapping("/students/{studentId}/courses")

public List<Course> getCourses(@PathVariable String studentId) {

return registrationService.fetchEnrolledCourses(user);

}

@GetMapping("/students/{studentId}/courses/{courseId}")

public Course retrieveDetailsForCourse(@PathVariable String studentId,

@PathVariable String courseId) {

return registrationService.fetchEnrolledCourses(user);

}

@Autowired

private MockMvc mockMvc;

@MockBean

private registrationService studentService;

Course mockCourse = new Course("Course1", "Spring", "10 Steps",

Arrays.asList("Learn Maven", "Import Project", "First Example",

"Second Example"));

String exampleCourseJson = "{\"name\":\"Spring\",\"description\":\"10 Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

@Test

public void retrieveDetailsForCourse() throws Exception {

Mockito.when(

studentService.retrieveCourse(Mockito.anyString(),

Mockito.anyString())).thenReturn(mockCourse);

RequestBuilder requestBuilder = MockMvcRequestBuilders.get(

"/students/Student1/courses/Course1").accept(

MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

System.out.println(result.getResponse());

String expected = "{id:Course1,name:Spring,description:10 Steps}";

// {"id":"Course1","name":"Spring","description":"10 Steps, 25 Examples and 10K Students","steps":["Learn Maven","Import Project","First Example","Second Example"]}

JSONAssert.assertEquals(expected, result.getResponse()

.getContentAsString(), false);

}

}

@Test

public void createStudentCourse() throws Exception {

Course mockCourse = new Course("1", "Smallest Number", "1",

Arrays.asList("1", "2", "3", "4"));

// studentService.addCourse to respond back with mockCourse

Mockito.when(

studentService.addCourse(Mockito.anyString(),

Mockito.any(Course.class))).thenReturn(mockCourse);

// Send course as body to /students/Student1/courses

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/courses/1",

response.getHeader(HttpHeaders.LOCATION));

@Test

public List<Enrollment> viewClasses () throws Exception {

viewClasses Classes = new ViewClasses()

Mockito.when(

studentController.viewClasses(Mockito.getClasses(),

Mockito.any(Classes.class))).thenReturn(mockClasses);

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/classschedule/1",

response.getHeaders(HttpHeaders.LOCATION));

@Test

Public List<Grade> viewGrades() throws Exception {

viewGrades Grades = new viewGrades()

Mockito.when(

studentController.viewGrades(),

Mockito.any(Grades.class))).thenReturn(mockGrades);

// Send course as body to /students/Student1/courses

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus())

@Test

public List<Course> getAll () throws Exception {

getAll Classes = new ()

Mockito.when(

studentController.getAll(Mockito.getClasses(),

Mockito.any(Classes.class))).thenReturn(mockClasses);

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/classschedule/1",

response.getHeaders(HttpHeaders.LOCATION));

@Test

public void delete(@pathVariable String coursed) throws Exception {

delete courseId = new delete()

Mockito.when(

studentController.delete(Mockito.delete()),

Mockito.any(courseId.class))).thenReturn(mockClasses);

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

@Test

public void update(@RequestBody Course course) throws Exception {

update Course = new update()

Mockito.when(

studentController.update(Mockito.getCourse(),

Mockito.any(Course.class))).thenReturn(mockClasses);

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/classschedule/1",

response.getHeaders(HttpHeaders.LOCATION));

}

}

**3.1.3 Test analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Requirement** | **Test Priority** | **Test Steps & pre-conditions** | **Expected Result** | **Obtained Result** | **Success/ Failure** | **Name of Tester** |
| **1** | Add course by admin | High | Just click on the class schedule button | A popup should be displayed that course added successfully | Displays the popup | Success | Akhila |
| **2** | Delete the course by admin | High | Just press the view grades tab | A popup should be displayed that course is displayed | Displays the popup | Success | Sreekanth |
| **3** | Check maximum strength | High | Enroll to the class | Get enrolled if class is not full else add to the waiting list | Course get enrolled as class is not full | Success | Akhila |
| **4** | Update Grades | High | Student should select the view grades | Should display the completed course grades | Displays the grades of completed courses | Success | Meghana |
| **5** | Update fees | High | Amount should be greater than 0 | Should display the updated fees | Displays the updated fee amount | Success | Varshini |
| **6** | Notify students through mail on enrollment changes | High | Open the mail box | A mail should be received on any updation on courses | An email is received on course professor change | Success | Sreekanth |

**4. CONTRIBUTIONS**

**4.1 Requirements**

|  |  |
| --- | --- |
| **Contributions** | **Developer Name** |
| **25%(fixed previous bugs)** | Sreekanth Vobilishetty |
| **25%(Course Edit feature)** | Akhila Yarlagadda |
| **25%(documentation of report, test cases)** | Piyusha Varshini Tirukovalluru |
| **25%(report and implementing the Test cases)** | Meghana Reddy Akkati |

**5. INSTALLATION INSTRUCTIONS**

Open spring tool suite on your device, select course registration and make a right click, a dropdown is displayed with the list, then select run as option to run the application as spring Boot App.

For the Front end execution, Go to angular project directory and type in below command

ng serve --proxy-config proxy.conf.json

Then open browser and hit <http://localhost:4200>

**6. PEER FEEDBACK SESSION**

In the code inspection session, we were given a positive feedback.