**DEVELOPMENT PHASE II 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

11/04/2019

**1. COURSE DETAIL REQUIREMENTS AND PAYMENT FEATURES**

Course Details feature enables the user to view the courses they enrolled for the semester in the My Classes tab which includes three internal features namely Class Schedule, View grades and Mandatory courses. Payment features enables the users to make payments for their tuition fee which includes paying tuition for the current semester and fines. It is present in the Finance tab which has the following features Make payment, View Account balance and View Payment history.

* 1. **My Classes tab**

It has features including viewing class schedule for the required semester, view grades secured for the particular semester and view the mandatory courses for the particular major of the user.

* + 1. **Class Schedule**

This feature enables the user to view their Schedule for the classes they enrolled depending upon the selection of the semester. User has no rights to modify the page.

* + 1. **View Grades**

This feature enables the user to view their grades for the courses depending on the selection of the semester. User has no rights to modify the page.

* + 1. **Mandatory courses**

This feature allows users to view the list of mandatory courses they have for their major. It gives the information about how many mandatory courses they have enrolled so far and how many are yet to be enrolled depending on the university policy.

* 1. **Finance tab**

This tab has features related to Payments which allows users to make payments for the semester in which they are enrolled. It gives the information regarding the Account balance and transaction history of the past payments.

A confirmation email is sent to the user after making the payment.

* + 1. **Make Payment**

This feature allows users to make payments which are due. It shows the Account balance and how much is due for them to make payment.

* + 1. **Past Payments**

This feature allows users to view their past payments they have made so far.

**1.3 Development phase II:**

In this phase we have developed the My classes and Finance features which allows users to view their class schedule for the semester, view grades and mandatory courses which are ought to be taken. Finance features includes making payments and viewing the payment history of the user. we have implemented the code for the following functional requirements

My Classes tab

View Class schedule

View grades

Mandatory courses

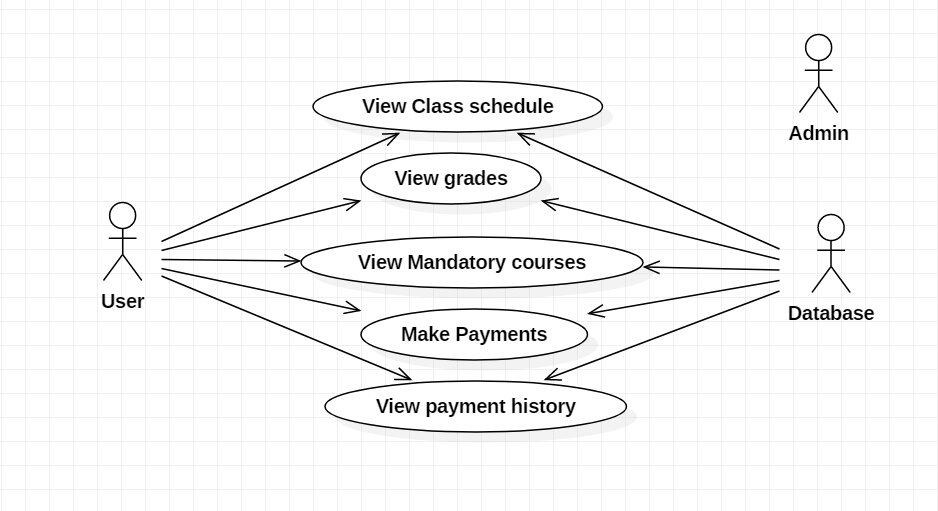
Finance tab

Make payment

Past payments

**2. UML DIAGRAMS**

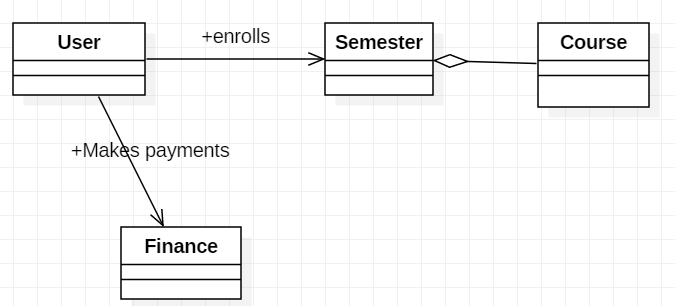
**2.1 Use case Diagram**

****

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 View Class schedule, view grades, view mandatory courses, make payments and view payment history.

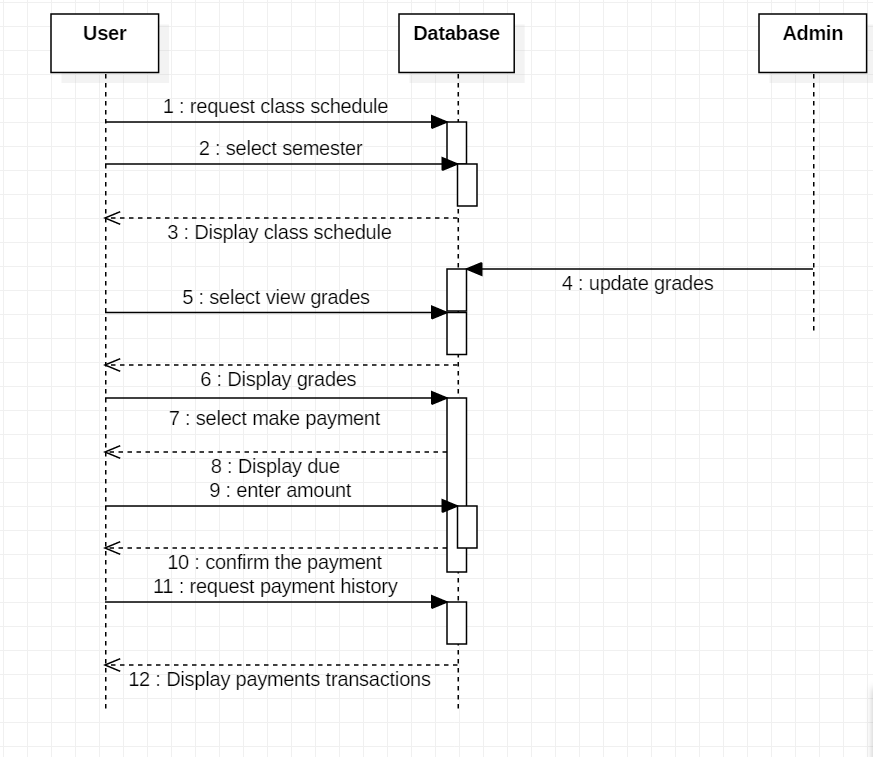
The User has a direct association with all the use cases mentioned in the diagram. The admin has no direct association for any of the use cases mentioned whereas database is directly associated with all the use cases since all the data needs to be stored and accessed from the database itself.

**2.2 Class Diagram**

****

The above diagram indicates 4 classes namely User, Semester, Course and Finance where the user enrolls with the semester and views the courses which is directly associated and User is enabled to make payments with the Finance class which are directly associated with each other.

**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<Payments> pastPayments() throws Exeception {

pastPayments payments new pastPayments()

Mockito.when(

studentController.payPayments(),

Mockito.any(payments.class))).thenReturn(mockpayments);

// 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()}

**3.1.3 Test analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Requirement** | **Test Priority** | **Test Steps & pre-conditions** | **Expected Result** | **Obtained Result** | **Success/ Failure** | **Name of Tester** |
| **1** | Class schedule has to be displayed | High | Just click on the class schedule button | A class schedule is displayed | The required class schedule page is displayed | Success | Akhila |
| **2** | View the grades of the enrolled courses | High | Just press the view grades tab | Grades of enrolled courses is displayed | The grades of the courses is displayed | Success | Sreekanth |
| **3** | If no classes enrolled for the current semester | High | Select the class schedule button | No class schedule should be displayed | An error message of no courses are registered for this semester | Success | Akhila |
| **4** | Make the fee payment | High | Select the make payment tab | Directs you to a page where due amount and enter amount is displayed | The required page as expected is displayed | Success | Meghana |
| **5** | Show the payment history | High | Just click on the payment history | Should display the past payments | The Payments made in the past is displayed | Success | Varshini |
| **6** | View the due amount of the fee | High | Just click on the make payments option | Displays the due amount | It displays the balance due | Success | Sreekanth |

**4. CONTRIBUTIONS**

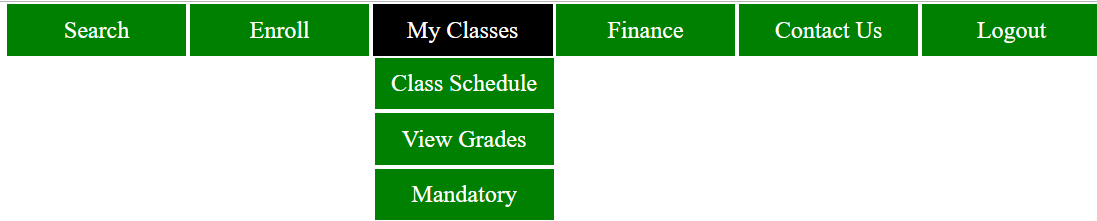
**4.1 Requirements**

|  |  |
| --- | --- |
| **Contributions** | **Developer Name** |
| **25%(payment feature and Testing)** | Sreekanth Vobilishetty |
| **25%(Class schedule, Mandatory courses and Testing)** | Akhila Yarlagadda |
| **25%(View grades, Testing and documentation of report)** | Piyusha Varshini Tirukovalluru |
| **25%(Payment feature and implementing the Test cases)** | Meghana Reddy Akkati |

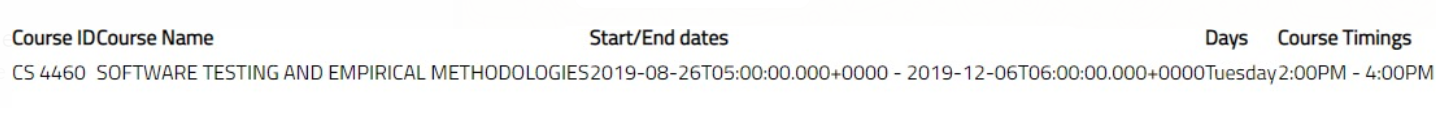
**5. USER MANUAL**

After the User enrolls his classes for the current semester, He/she can view their current class, their grades for all the classes they have enrolled for so far and list of mandatory classes specified for the particular major of the user.

**My Classes tab:**

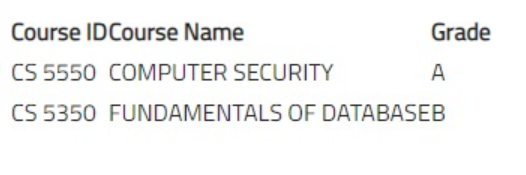


**View Class Schedule:**



In the figure, we see the class schedule for the current semester which has Course ID, Class name, Start and end dates, Days and Timings of the class.

**View Grades:**



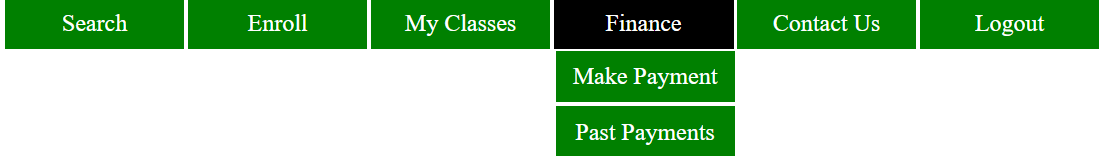
In the figure, we can view the grades secured by the user for all the courses enrolled so far.

**Mandatory courses:**

It displays the list of all the mandatory courses for the user depending upon the major to which they belong. It shows the one’s which are completed and the one’s which are yet to be registered.

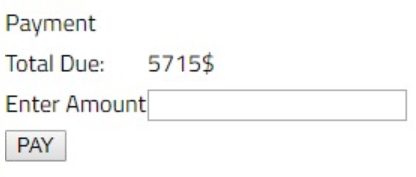
**Finance tab:**

This tab enables the user to make payments and view their past payments.

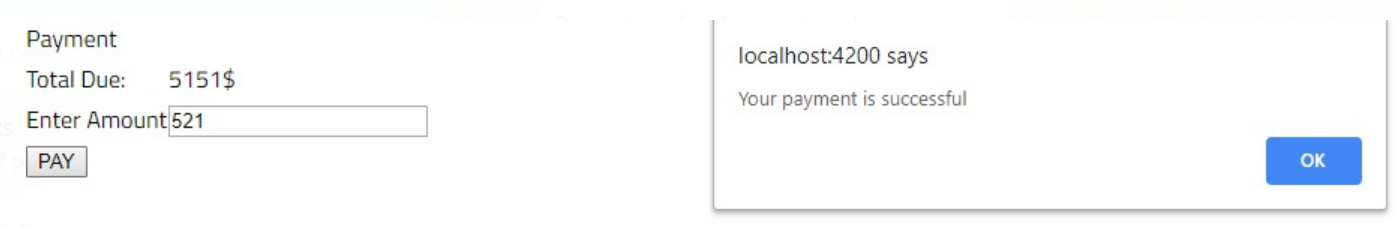
****

**Make payment:**

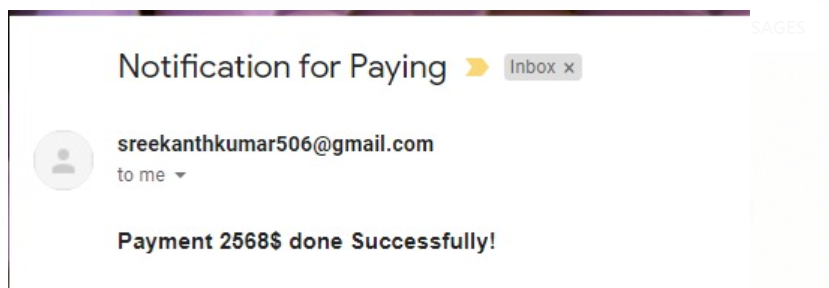
In the figure below, we are able to view the total due amount that needs to be paid and asks us to enter the amount that user wants to pay.



The Figure below shows that the payment made is successful and the amount is due amount is changed accordingly.



We get a confirmation email saying that the payment is successful.



**Past Payments:**

In this figure, it shows the payment details and history of the payments made which includes the Payment ID, Payment date and the amount paid.



**6. 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>

**7. PEER FEEDBACK SESSION**

In the code inspection session, we were given a constructive feedback which said to comment every function and classes used in the code for a better understanding.