**PROFESSOR SCHEDULE MANAGEMENT SYSTEM (PMS)**

**SPECIFICATION-BASED TEST DOCUMENT**

**Prepared by Team 5**

Alejandro Perez

David Rocha

Raul Espinosa

Samuel Yorizzo

**Prepared For**

Professor Peter Clarke

Fundamentals of Software Testing, CEN4072 - U01

Florida International University, Spring 2019

February 22nd, 2019

**Abstract:**

This document contains the specification-based testing techniques applied to the Professor Schedule Management System (PMS) software. Testing applied to the software serves as validation of its intended use and is based on the use cases outlined in the software documentation and features implemented by the software designers.

Outlined in the document is the system overview, system requirements and testing methodology. Procedures and documentation are included for testing done using JUnit, Mockito and Rational Functional Tester in the Eclipse IDE. Test cases and results are documented for all unit, subsystem and system test conducted by the testing team.

Table of Contents

**Abstract1**

**Table of Contents2**

**1. Introduction 3**

1.1 Overview of System 3

1.2 Requirements of the System 3

1.2.1 Functional Requirements 3

1.2.2 Non-Functional Requirements 7

1.3 Overall Testing Approach 7

1.3.1 Unit Testing Approach 9

1.3.2 Subsystem Testing Approach 9

1.3.4 System Testing Approach 9

1.4 Terminology 9

1.5 Document Organization 10

**2. Specification Test Plan 11**

2.1 Organization 11

2.2 Hardware and Software Requirements 12

2.3 Test References Items 13

2.4 Tested Features 13

2.5 Features Not Tested 13

2.6 Work Breakdown 14

**3. Unit Testing 15**

3.1 Unit Test Cases 15

3.1.1 Test Identification and Objectives 15

3.1.2 Test Criteria and Procedures 21

3.1.3 Test Cases 21

3.2 Actual Test Results 62

**4. Subsystem Testing 66**

4.1 Subsystem Test Cases 66

4.1.1 Test Identification and Objectives 66

4.1.2 Test Criteria and Procedures 67

4.1.3 Test Cases 68

4.2 Actual Test Results 71

**5. System Testing 72**

5.1 Subsystem Test Cases 72

5.1.1 Test Identification and Objectives 72

5.1.2 Test Criteria and Procedures 73

5.1.3 Test Cases 74

5.2 Actual Test Results 83

**6. Test Summary Report 84**

**7. Risk and Contingencies 87**

**8. Approvals 88**

**9. Glossary 89**

**10. Appendix 90**

10.1 Appendix A 90

10.2 Appendix B 90

10.3 Appendix C91

10.4 Appendix D 101

10.5 Appendix E 103

**1. Introduction**

This chapter contains an overview of the system being tested, its requirements, as well as documentation on the testing approach, testing terminology, and overall organization of the document.

The software being tested, Professor Schedule Management System, is henceforth referred as PMS throughout the length of this document. All references to PMS encompass the entire original artifact being tested.

**1.1 Overview of System**

PMS software was designed with the intent to provide Florida International University professors with a tool to add, manage, and track class schedules. At the time of the software’s conception, no tool was available to provide this functionality, and this served as the motivation behind the creation of the program.

The intended functionality of the software would provide professors with an easily accessible interface to add class schedules, identify schedule conflicts, manage and edit current schedules, and provide time alerts during classes.

**1.2 Requirements of the System**

This section contains the full list of system requirements for the PMS software. Section 1.2.1 contains the functional requirements, and section 1.2.2 contains the non-functional requirements. All requirements presented are elicited from the original PMS software documentation.

**1.2.1 Functional Requirements**

Functional requirements are taken from the documentation provided by the software designers and are displayed with the use case identification and description.

|  |  |
| --- | --- |
| PMS\_001 – Login | * The system shall provide the user with a login template. * The system shall take the user’s username and password. * The system shall validate the user’s username and password. * The system shall allow or deny access to the user. |
| PSM\_002 – Logout | * The system shall provide the user with a logout template. * The system shall process logout request by the user. * The system shall provide the option to cancel logout process. |
| PSM\_003 – Security | * The system shall provide the user with a template to manage idle auto logout. * The system shall provide option to select best feature according to user’s preference. * The system shall provide the option to cancel request process. |
| PSM\_004 – Schedule Setup | * The system shall provide the user with a schedule setup template. * The system shall process user’s entered data. * The system shall validate user’s entered data. * The system shall check for conflicts in the schedule. * The system shall save the new schedule. * The system shall provide the option to cancel schedule setup process. |
| PSM\_006 - Alarms | * The system shall allow the users to set up timing related alarms with different messages and sounds. |
| PSM\_007 – Custom Warning | * The system shall allow the users to set up custom warnings |
| PSM\_008 – Message Popup | * The system shall warn the user about the current situation by displaying a pop-up message and audible sound. |
| PSM\_009 – Program Priority | * The system shall give warnings and pop-ups high priority over other programs. |
| PSM\_010 – Exam Set-up | * The system shall provide the user with a template to schedule exams. * The system shall process user’s entered data. * The system shall allow the user to set up warnings for exams. * The system shall save the changes made. * The system shall display current time and time left. * The system shall display warnings and notifications * The system shall provide the option to cancel exam set-up process. |
| PSM\_011 – Presentation setup | * The system shall provide the user with a presentation setup template. * The system shall process the user’s entered data. * The system shall record changes made. * The system shall display team’s name and time left. * The system shall display a notification when the time for presentation has ended. * The system shall display time until next presentation starts. * The system shall provide the option to cancel presentation setup process. |
| PSM\_012 – Edit Schedule | * The system shall provide the user with a template to edit schedules. * The system shall process the user’s entered data. * The system shall validate the user’s entered data. * The system shall check for conflicts in the schedule. * The system shall save the edited schedule. * The system shall provide the option to cancel edit schedule process. |
| PSM\_013 – Remove Schedule | * The system shall provide the user with a template to remove schedules * The system shall display confirmation message before removing a schedule * The system shall remove the schedule data. * The system shall provide the option to cancel schedule remove process. |
| PSM\_014 – End of Semester Schedule Clear | * The system shall provide the user with a template to input first schedule. * The system shall allow the user to input the last day of schedule * The system shall compare the data with professor’s last day. Later days will get priority. * The system shall provide the user the option to modify last day of schedule. |
| PSM\_015 – Data Validation | * The system shall verify every field of inputted data is correct * The system shall notify the user when data is incorrect or wrong format. |
| PSM\_016 – Schedule Conflicts | * The system shall verify every field of inputted data format * The system shall notify the user when data is incorrect or wrong format |
| PSM\_017 – Password Conflicts | * The system shall verify every field of inputted data is correct * The system shall give an error to user letting know the password is not correct or does not match the system. * The system shall enable the use of password problem button * The system shall open a new page letting user answer a question in or der to reset password. |
| PSM\_018 – Custom Feature | * The system shall provide the user with a template for features. * The system shall display the features selection of options and tab menus. |
| PSM\_019 – Late Notice for Professor | * The system shall check login time with current. * The system shall notify professor when he/she is late. |
| PSM\_020 – Single Day | * The system shall display a menu that the user can enter data. * The system shall create a single day function to apply data. * The system shall display a window to confirm creation of single day with inputted data. * The system will allow the user to confirm creation of single day. |
| PSM\_021 – Professor Attendance | * The system shall check the time and date that the login occurred and store in records. * The system shall check to see if during the previous log in to the most recent if the professor has missed any classes during the period. * The system shall notify the professor the dates he was missing from class and the closes to continue normal system run time. |
| PSM\_005 – Help Password | * The system shall validate username * The system shall provide the user with a security password template. * The system shall validate user’s secret question. * The system shall provide a new password template for user. * The system shall validate new password. * The system shall save new password. |

**1.2.2 Non-Functional Requirements**

The system is specifically design for Florida International University (henceforth referred to as FIU) professors and all design features and implementations were created based on this premise. All features are specific to the FIU professors needs and FIU’s system architecture.

Organizational structure of the software must be compatible with FIU’s scheduling system.

PSM’s scheduling setup and overall features must be dependable for professor’s usage.

PSM’s performances must be reliable and predictable for professor’s usage.

PSM must be available and compatible with every professor’s system.

**1.3 Overall Testing Approach**

This section contains information on the overall testing approach used to validate PSM’s implementation against the use cases outlined in section 1.2.

The testing of the program is done using specification-based testing techniques and all testing is done following the specifications outlined in the requirements. Testing follows the program’s three tier architecture (interface tier, logic tier and data tier) to test each tier separately and together.

The selection of the test set for test cases is done through equivalence partitioning and boundary analysis. Furthermore, in order to test the tiers thoroughly, testing is done in three categories: Unit testing, subsystems testing and systems testing. Following information shows further details on each of these subdivisions of testing.

A close up of a sign

Description automatically generatedFigure 1.3.1 Test Plan Flowchart.

**1.3.1 Unit Testing Approach**

Unit testing is conducted on two specific classes of the PSM software, the appController class and the DBConnection class. The classes are isolated from the system and states, variable and methods are tested independent of the rest of the system. Test cases are created using equivalence partitioning and boundary analysis. Tests are conducted in the Eclipse environment using JUnit and Mockito testing tools.

**1.3.2 Subsystem Testing Approach**

Subsystem testing is conducted on the logic package of the PSM software. The package is isolated from the rest of the system and tested independently from others. Dependency classes and calls are mocked to control input and output for testing purposes. Test cases are created using equivalence partitioning and boundary analysis. Tests are conducted in the Eclipse environment using JUnit and Mockito testing tools.

**1.3.3 System Testing Approach**

System testing is conducted on the entire PSM software. Test cases are created using equivalence partitioning and boundary analysis and validated against the software’s use cases. Test are conducted in the Eclipse environment using IBM’s Rational Functional Tester.

**1.4 Terminology**

|  |  |
| --- | --- |
| PMS | Professor Schedule Management System |
| JUnit | Testing framework used for unit and subsystem testing. Creates and calls test cases. |
| Mockito | Testing framework used for unit and subsystem testing. Creates mocks classes to use in testing. |
| RFT | Rational Functional Tester. Testing framework used in system testing. Automated system testing that mimics the actions of user. |
| Eclipse IDE | Java environment used to create and run the software and tests. |
| FIU | Florida International University |
| SQL | Relational Database System |

**1.5 Document Organization**

This section explains the organization and characteristics of the documentation in the following chapters and sections. Beyond this section, the following 5 chapters will contain documentation relating to the testing of the PSM software.

Chapter 2 explains the initial setup and preparations for testing, including details on the roles of testing members, test planning, test scheduling, hardware and software used for testing, and features tested and not tested.

Chapters 3, 4 and 5 detail test case documentation for unit testing, subsystem testing and system testing respectively.

Chapter 6 contains the summary report of the test cases documented in sections 3, 4 and 5.

Chapter 7 contains risks and contingencies encountered and analyzed in the process of testing.

**2. Specification Test Plan**

The purpose of this chapter is to specify the team’s plan for testing in the project, including each member’s roles, project software and hardware requirements, reference materials, features tested and not tested, and to give a breakdown of the work for phase 1.

**2.1 Organization**

The following table illustrates the team’s roles and tasks performed during phase 1 of the project.

|  |  |  |
| --- | --- | --- |
| **Team 5 – Phase 1** | | |
| **Member Name** | **Member Role** | **Tasks performed** |
| Raul Espinosa | Team Leader | Unit Testing (Backup), Subsystem Testing (Backup) |
| Alejandro Perez | Minute Taker | Unit Testing (Lead) |
| Samuel Yorizzo | Time Keeper | Systems Testing (Lead), Subsystem Testing (Backup) |
| David Rocha |  | Subsystem Testing (Lead), System Testing (Backup) |

**2.2 Hardware and Software Requirements**

The following hardware is used to write and execute the code for the tests:

**Hardware Requirements:**

**Machine 1 (laptop)**

Processor: 2.8GHz dual-core Intel core i7

RAM 16GB

**Machine 2 (laptop)**

1.6GHz Intel core i5

RAM 4GB

**Machine 3 (laptop)**

3.1GHz Intel core i5

RAM 8GB

**Machine 4 (laptop)**

2.4GHz Intel core i5

RAM 8GB

**Software Requirements:**

* Eclipse Oxygen.3a Release (4.7.3a)
* Google Chrome Browser Version 72.0.3626.109
* JUnit (version 4.12): Framework for unit testing
* Mockito (mockito-core 2.23.4): Framework for mocking.
* MySQL (version 8.0.15): database management system
* RFT (version 9.2.1): Framework for system testing.

**2.3 Test References Items**

* JUnit (version 4.12): Framework used for unit and subsystem testing.
* Mockito (mockito-core 2.23.4): Framework used for mocking.
* MySQL (version 8.0.15): Database management system.
* RFT (version 9.2.1): Framework used for system testing.
* Eclipse Oxygen.3a Release (4.7.3a): IDE used for testing.
* Professor Schedule Manager (PSM) User’s Guide PDF: Manual for the PSM application.
* Professor Schedule Manager (PSM) FP (authors: Jessica Canonico, Peter Wong, Kurt Keipert, Luis Rizo, Marcos Rojas): Final Paper for the application.

**2.4 Tested Features**

The following functionalities were tested. Please note that in the PSM FP and User’s Guide, some of the Use Cases (namely the non-tested features and Database access) did not include numbered PSM prefixes.

PSM\_001-Login

PSM\_002-Logout

PSM\_003-Authenticate

PSM\_004-Schedule Setup

PSM\_012-Schedule Edit

PSM\_008-Message Pop-up

Database access

**2.5 Features Not Tested**

Password Idle

End Semester Cleanup

Presentation Setup

Exam Setup

Single Day Setup

Features

Alarm

**2.6 Work Breakdown**

The following table illustrates the work breakdown for phase 1 of the project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Work Breakdown – Phase 1** | | | | |
| **ID** | **Task Name** | **Assignee** | **Date Started** | **Date Ended** |
| 1 | PSM Test Schedule | All | 1/9/19 | 1/19/19 |
| 2 | Review Program Documents | All | 1/16/19 | 1/23/19 |
| 3 | Download Tools | All | 1/23/19 | 1/25/19 |
| 4 | Create DB / Learn tools | All | 1/25/19 | 1/29/19 |
| 5 | Begin Unit Testing | Alejandro, Raul | 1/29/19 | 2/19/19 |
| 6 | Begin Subsystem Testing | David | 1/29/19 | 2/19/19 |
| 7 | Begin System Testing | Samuel | 1/29/19 | 2/19/19 |

**3. Unit Testing**

**3.1 Unit Test Cases**

In this section, we present the test cases for unit testing. They are identified by id and purpose. Each test case ID is written using a unique prefix and a running counter for the test cases per project instructions. Test cases are divided between unit tests of appController class and unit tests of DBConnections class.

appController tests cases go from counter 001 to counter 030, showing 30 test cases for the appCopntroller class.

DBConnection tests cases go from counter 031 to counter 076, showing 45 test cases for the DBConnection class.

Each method tested has a sunny day test case and a rainy day test case to test expected output and not expected output.

Test set for test cases are chosen using equivalence partitioning for legal and illegal inputs. Inputs are further selected using boundary analysis to test inputs close enough to legal values to see how the system handled them.

**3.1.1 Test Identification and Objective**

Test Identification with test IDs and Purpose. Starts in following page.

**appController Unit Testing Test Cases**

|  |  |
| --- | --- |
| **Test ID** | **Purpose** |
| Unit\_Test\_checkClear\_Sunny\_001 | Test if the checkClear method correctly tests the dates against acceptable dates |
| Unit\_Test\_checkClear\_Rainy\_002 | Test if the checkClear method correctly tests the dates against acceptable dates |
| Unit\_Test\_checkTimes\_Sunny\_003 | Test if the checkTimes method correctly checks the times in the schedule for a specific course id |
| Unit\_Test\_checkTimes\_Rainy\_004 | Test if the checkTimes method correctly checks the times in the schedule for a specific course id |
| Unit\_Test\_getData\_Sunny\_005 | Test if the getData method retrieves data from database and stores it in variables |
| Unit\_Test\_getData\_Rainy\_006 | Test if the getData method retrieves data from database and stores it in variables |
| Unit\_Test\_LogIn\_Sunny\_007 | Test if the LogIn method successfully logs into the database with specific username/password |
| Unit\_Test\_LogIn\_Rainy\_008 | Test if the LogIn method successfully logs into the database with specific username/password |
| Unit\_Test\_getCon\_Sunny\_009 | Test if the getCon method successfully retrieves the database instance from class |
| Unit\_Test\_getCon\_Rainy\_010 | Test if the getCon method successfully retrieves the database instance from class |
| Unit\_Test\_timerParser\_Sunny\_011 | Test if the timerParser method correctly gets a time string from database and parses it into int variables of class |
| Unit\_Test\_timerParser\_Rainy\_012 | Test if the timerParser method correctly gets a time string from database and parses it into int variables of class |
| Unit\_Test\_dateParser\_Sunny\_013 | Test if the dateParser method correctly gets a date string from database and parses it into int variables of class |
| Unit\_Test\_dateParser\_Rainy\_014 | Test if the timerParser method correctly gets a time string from database and parses it into int variables of class |
| Unit\_Test\_ReturnHr\_Sunny\_015 | Test if the ReturnHr method correctly retrieves the hr variable from class |
| Unit\_Test\_ReturnHr\_Rainy\_016 | Test if the ReturnHr method correctly retrieves the hr variable from class |
| Unit\_Test\_ReturnMin\_Sunny\_017 | Test if the ReturnMin method correctly retrieves the min variable from class |
| Unit\_Test\_ReturnMin\_Rainy\_018 | Test if the ReturnMin method correctly retrieves the min variable from class |
| Unit\_Test\_getEndTime\_Sunny\_019 | Test if the getEndTime method correctly checks current time against the scheduled end time of class |
| Unit\_Test\_getEndTime\_Rainy\_020 | Test if the getEndTime method correctly checks current time against the scheduled end time of class |
| Unit\_Test\_getSemesterClear\_Sunny\_021 | Test if the setSemesterClear method correctly gets the new date to clear semester from class |
| Unit\_Test\_getSemesterClear\_Rainy\_022 | Test if the setSemesterClear method correctly gets the new date to clear semester from class |
| Unit\_Test\_setSemesterClear\_Sunny\_023 | Test if the getSemesterClear method correctly sets the new date to clear semester from class |
| Unit\_Test\_setSemesterClear\_Rainy\_024 | Test if the gettSemesterClear method correctly sets the new date to clear semester from class |
| Unit\_Test\_get15BeforeEnd\_Sunny\_025 | Test if the get15BeforeEnd method correctly gets the time 15 mins before the end of schedule class |
| Unit\_Test\_get15BeforeEnd\_Rainy\_026 | Test if the get15BeforeEnd method correctly gets the time 15 mins before the end of schedule class |
| Unit\_Test\_get5BeforeEnd\_Sunny\_027 | Test if the get 5BeforeEnd method correctly gets the time 5 mins before the end of schedule class |
| Unit\_Test\_get5BeforeEnd\_Rainy\_028 | Test if the get 5BeforeEnd method correctly gets the time 5 mins before the end of schedule class |
| Unit\_test\_loggedin\_Sunny\_029 | Test if the loggedin method, containing the login logic for main, correctly performs login operations |
| Unit\_test\_loggedin\_Rainy\_030 | Test if the loggedin method, containing the login logic for main, correctly performs login operations |

|  |  |
| --- | --- |
| **DBConnection Unit Test** |  |
| Test ID | Purpose |
| Unit\_Test\_connect\_Sunny\_031 | Test if the connect method correctly stablishes a connection with the database |
| Unit\_Test\_connect\_Rainy\_032 | Test if the connect method correctly stablishes a connection with the database |
| Unit\_Test\_disconnect\_Sunny\_033 | Test if the disconnect method correctly disconnects from the database |
| Unit\_Test\_disconnect\_Rainy\_034 | Test if the disconnect method correctly disconnects from the database |
| Unit\_Test\_fetchCourseID\_Sunny\_035 | Test if the fetchCourseID method gets the course id from the database given a specific course id parameter |
| Unit\_Test\_fetchCourseID\_Rainy\_036 | Test if the fetchCourseID method gets the course id from the database given a specific course id parameter |
| Unit\_Test\_getEndDates\_Sunny\_037 | Test if the getEndDates method gets the end gates of a semester from the database given a specific course id |
| Unit\_Test\_getEndDates\_Rainy\_038 | Test if the getEndDates method gets the end gates of a semester from the database given a specific course id |
| Unit\_Test\_getCourses\_Sunny\_039 | Test if the getCourses method returns a list with all the course id of all the courses in the database |
| Unit\_Test\_getCourses\_Rainy\_040 | Test if the getCourses method returns a list with all the course id of all the courses in the database |
| Unit\_Test\_fetchCourseSubj\_Sunny\_041 | Test if the fetchCourseSubj method returns the course subject from the database given a specific course id |
| Unit\_Test\_fetchCourseSubj\_Rainy\_042 | Test if the fetchCourseSubj method returns the course subject from the database given a specific course id |
| Unit\_Test\_fetchCourseName\_Sunny\_043 | Test if the fetchCourseName method returns the course name from the database given a specific course id |
| Unit\_Test\_fetchCourseName\_Rainy\_044 | Test if the fetchCourseName method returns the course name from the database given a specific course id |
| Unit\_Test\_fetchCourseSemester\_Sunny\_045 | Test if the fetchCourseSemester method returns the semester from the database given a specific course id |
| Unit\_Test\_fetchCourseSemester\_Rainy\_046 | Test if the fetchCourseSemester method returns the semester from the database given a specific course id |
| Unit\_Test\_fetchCourseStart\_Sunny\_047 | Test if the fetchCourseStart method returns the start date of semester from the database given a specific course id |
| Unit\_Test\_fetchCourseStart\_Rainy\_048 | Test if the fetchCourseStart method returns the start date of semester from the database given a specific course id |
| Unit\_Test\_fetchCourseEnd\_Sunny\_049 | Test if the fetchCourseEnd method returns the end date of semester from the database given a specific course id |
| Unit\_Test\_fetchCourseEndt\_Rainy\_050 | Test if the fetchCourseEnd method returns the end date of semester from the database given a specific course id |
| Unit\_Test\_fetchStartMon\_Sunny\_051 | Test if the fetchStartMon method correctly returns the time a course starts on Monday given a specific course id |
| Unit\_Test\_fetchStartMon\_Rainy\_052 | Test if the fetchStartMon method correctly returns the time a course starts on Monday given a specific course id |
| Unit\_Test\_fetchEndMon\_Sunny\_053 | Test if the fetchEndtMon method correctly returns the time a course ends on Monday given a specific course id |
| Unit\_Test\_fetchEndMon\_Rainy\_054 | Test if the fetchEndMon method correctly returns the time a course ends on Monday given a specific course id |
| Unit\_Test\_fetchStartTue\_Sunny\_055 | Test if the fetchStartTue method correctly returns the time a course starts on Tuesday given a specific course id |
| Unit\_Test\_fetchStartTue\_Rainy\_056 | Test if the fetchStartTue method correctly returns the time a course starts on Tuesday given a specific course id |
| Unit\_Test\_fetchEndTue\_Sunny\_057 | Test if the fetchEndTue method correctly returns the time a course ends on Tuesday given a specific course id |
| Unit\_Test\_fetchEndTue\_Rainy\_058 | Test if the fetchEndTue method correctly returns the time a course ends on Tuesday given a specific course id |
| Unit\_Test\_fetchStartWed\_Sunny\_059 | Test if the fetchStartWed method correctly returns the time a course starts on Wednesday given a specific course id |
| Unit\_Test\_fetchStartWed\_Rainy\_060 | Test if the fetchStartWed method correctly returns the time a course starts on Wednesday given a specific course id |
| Unit\_Test\_fetchEndWed\_Sunny\_061 | Test if the fetchEndWed method correctly returns the time a course ends on Wednesday given a specific course id |
| Unit\_Test\_fetchEndWed\_Rainy\_062 | Test if the fetchEndWed method correctly returns the time a course ends on Wednesday given a specific course id |
| Unit\_Test\_fetchStartThu\_Sunny\_063 | Test if the fetchStartThu method correctly returns the time a course starts on Thursday given a specific course id |
| Unit\_Test\_fetchStartThu\_Rainy\_064 | Test if the fetchStartThu method correctly returns the time a course starts on Thursday given a specific course id |
| Unit\_Test\_fetchEndThu\_Sunny\_065 | Test if the fetchEndThu method correctly returns the time a course ends on Thursday given a specific course id |
| Unit\_Test\_fetchEndThu\_Rainy\_066 | Test if the fetchEndhu method correctly returns the time a course ends on Thursday given a specific course id |
| Unit\_Test\_fetchStartFri\_Sunny\_067 | Test if the fetchStartFri method correctly returns the time a course starts on Friday given a specific course id |
| Unit\_Test\_fetchStartFri\_Rainy\_068 | Test if the fetchStartFri method correctly returns the time a course starts on Friday given a specific course id |
| Unit\_Test\_fetchEndFri\_Sunny\_069 | Test if the fetchEndFri method correctly returns the time a course ends on Friday given a specific course id |
| Unit\_Test\_fetchEndFri\_Rainy\_070 | Test if the fetchEndFri method correctly returns the time a course ends on Friday given a specific course id |
| Unit\_Test\_fetchStartSat\_Sunny\_071 | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Unit\_Test\_fetchStartSat\_Rainy\_072 | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Unit\_Test\_fetchEndSat\_Sunny\_073 | Test if the fetchEndSat method correctly returns the time a course ends on Saturday given a specific course id |
| Unit\_Test\_fetchEndSat\_Rainy\_074 | Test if the fetchEndSat method correctly returns the time a course ends on Saturday given a specific course id |
| Unit\_Test\_storeClassInfo\_Sunny\_075 | Test if the storeClassInfo method correctly stores the class info into database. |
| Unit\_Test\_storeClassInfo\_Rainy\_076 | Test if the storeClassInfo method correctly stores the class info into database. |

**3.1.2 Test Criteria and Procedures**

Test inputs vary from strings to integers to arrays and arraylists. Inputs are selected from the set created by equivalence partitioning and boundary analysis for the expected input on use cases.

Test procedure started by creating testing packages for each class to be tested, the appController class and the DBConnection class. Classes are imported from the PSM package to the test package and JUnit tests are made to create the testing methods.

Mockito is then used to mock classes in the testing methods and JUnit methods are used to test the entire classes.

**3.1.3 Test Cases**

Test case documentation.

**appController Unit Test**

|  |  |
| --- | --- |
| @Before  Mock setup | controller = new appController();    dbCon = mock(DBConnection.class);  statement = mock(Statement.class);  connection = mock(Connection.class);  authMock = mock(Authenticate.class);  icMock = mock(InterfaceController.class);  logMock = mock(LoginForm.class);  msgMock = mock(Messages.class);    controller.setDB(dbCon);  controller.setAuthenticate(authMock);  controller.setInterfaceController(icMock); |
| @After  Mock teardown | controller = null; |

Table 3.1 Mock setUp and tearDown called before and after mocks respectively.

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_checkClear\_Sunny\_001 |
| Test Purpose: | Test if the checkClear method correctly tests the dates against acceptable dates |
| Test Setup | ArrayList<String> date = new ArrayList<String>();  date.add("01/08/19");  date.add("01/08/19");  date.add("01/08/19");  when(dbCon.getEndDates()).thenReturn(date)  See Table 3.1 for setUp mocks. |
| Test Input: | controller.checkClear() |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_checkClear\_Rainy\_002 |
| Test Purpose: | Test if the checkClear method correctly tests the dates against acceptable dates |
| Test Setup | ArrayList<String> date = new ArrayList<String>();  date.add("03/08/19");  date.add("03/08/19");  date.add("03/08/19");  when(dbCon.getEndDates()).thenReturn(date)  See Table 3.1.3.1 for setUp mocks. |
| Test Input: | controller.checkClear() |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_checkTimes\_Sunny\_003 |
| Test Purpose: | Test if the checkTimes method correctly checks the times in the schedule for a specific course id |
| Test Setup | ArrayList<Integer> date = new ArrayList<Integer>();  date.add(1101);  defSub = "ENC";  defSemester = "Spring";  defCourseName = "English";  defCourseStart = "01/08/19";  defCourseEnd = "05/01/19";  defMonStart = "15:00";  defMonEnd = "16:00";  defTueStart = "15:00";  defTueEnd = "16:00";  defWedStart = "15:00";  defWedEnd = "16:00";  defThuStart = "15:00";  defThuEnd = "16:00";  defFriStart = "15:00";  defFriEnd = "16:00";  defSatStart = "15:00";  defSatEnd = "16:00";  when(dbCon.getCourses()).thenReturn(date);  when(dbCon.fetchCourseSubj(1101)).thenReturn(defSub);  when(dbCon.fetchCourseSemester(1101)).thenReturn(defSemester);  when(dbCon.fetchCourseName(1101)).thenReturn(defCourseName);  when(dbCon.fetchCourseStart(1101)).thenReturn(defCourseStart);  when(dbCon.fetchCourseEnd(1101)).thenReturn(defCourseEnd);  when(dbCon.fetchStartMon(1101)).thenReturn(defMonStart);  when(dbCon.fetchEndMon(1101)).thenReturn(defMonEnd);  when(dbCon.fetchStartTue(1101)).thenReturn(defTueStart);  when(dbCon.fetchEndTue(1101)).thenReturn(defTueEnd);  when(dbCon.fetchStartWed(1101)).thenReturn(defWedStart);  when(dbCon.fetchEndWed(1101)).thenReturn(defWedEnd);  when(dbCon.fetchStartThu(1101)).thenReturn(defThuStart);  when(dbCon.fetchEndThu(1101)).thenReturn(defThuEnd);  when(dbCon.fetchStartFri(1101)).thenReturn(defFriStart);  when(dbCon.fetchEndFri(1101)).thenReturn(defFriEnd);  when(dbCon.fetchStartSat(1101)).thenReturn(defSatStart);  when(dbCon.fetchEndSat(1101)).thenReturn(defSatEnd);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.checkTimes() |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_checkTimes\_Rainy\_004 |
| Test Purpose: | Test if the checkTimes method correctly checks the times in the schedule for a specific course id |
| Test Setup | ArrayList<Integer> date = new ArrayList<Integer>();  date.add(1101);  defSub = "";  defSemester = "";  defCourseName = "";  defCourseStart = "";  defCourseEnd = "";  defMonStart = "";  defMonEnd = "";  defTueStart = "";  defTueEnd = "";  defWedStart = "";  defWedEnd = "";  defThuStart = "";  defThuEnd = "";  defFriStart = "";  defFriEnd = "";  defSatStart = "";  defSatEnd = "";  when(dbCon.getCourses()).thenReturn(date);  when(dbCon.fetchCourseSubj(1101)).thenReturn(defSub);  when(dbCon.fetchCourseSemester(1101)).thenReturn(defSemester);  when(dbCon.fetchCourseName(1101)).thenReturn(defCourseName);  when(dbCon.fetchCourseStart(1101)).thenReturn(defCourseStart);  when(dbCon.fetchCourseEnd(1101)).thenReturn(defCourseEnd);  when(dbCon.fetchStartMon(1101)).thenReturn(defMonStart);  when(dbCon.fetchEndMon(1101)).thenReturn(defMonEnd);  when(dbCon.fetchStartTue(1101)).thenReturn(defTueStart);  when(dbCon.fetchEndTue(1101)).thenReturn(defTueEnd);  when(dbCon.fetchStartWed(1101)).thenReturn(defWedStart);  when(dbCon.fetchEndWed(1101)).thenReturn(defWedEnd);  when(dbCon.fetchStartThu(1101)).thenReturn(defThuStart);  when(dbCon.fetchEndThu(1101)).thenReturn(defThuEnd);  when(dbCon.fetchStartFri(1101)).thenReturn(defFriStart);  when(dbCon.fetchEndFri(1101)).thenReturn(defFriEnd);  when(dbCon.fetchStartSat(1101)).thenReturn(defSatStart);  when(dbCon.fetchEndSat(1101)).thenReturn(defSatEnd);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.checkTimes() |
| Expected Output: | True |
| Test ID: | Unit\_Test\_getData\_Sunny\_005 |
| Test Purpose: | Test if the getData method retrieves data from database and stores it in variables |
| Test Setup | defSub = "ENC";  defSemester = "Spring";  defCourseName = "English";  defCourseStart = "01/08/19";  defCourseEnd = "05/01/19";  defMonStart = "15:00";  defMonEnd = "16:00";  defTueStart = "15:00";  defTueEnd = "16:00";  defWedStart = "15:00";  defWedEnd = "16:00";  defThuStart = "15:00";  defThuEnd = "16:00";  defFriStart = "15:00";  defFriEnd = "16:00";  defSatStart = "15:00";  defSatEnd = "16:00";  String[] allValues = {defSub, defSemester, defCourseName, defCourseStart,defCourseEnd, defMonStart, defMonEnd, defTueStart,  defTueEnd, defWedStart, defWedEnd, defThuStart, defThuEnd, defFriStart, defFriEnd, defSatStart, defSatEn };  when(dbCon.fetchCourseSubj(1101)).thenReturn(defSub);  when(dbCon.fetchCourseSemester(1101)).thenReturn(defSemester);  when(dbCon.fetchCourseName(1101)).thenReturn(defCourseName);  when(dbCon.fetchCourseStart(1101)).thenReturn(defCourseStart);  when(dbCon.fetchCourseEnd(1101)).thenReturn(defCourseEnd);  when(dbCon.fetchStartMon(1101)).thenReturn(defMonStart);  when(dbCon.fetchEndMon(1101)).thenReturn(defMonEnd);  when(dbCon.fetchStartTue(1101)).thenReturn(defTueStart);  when(dbCon.fetchEndTue(1101)).thenReturn(defTueEnd);  when(dbCon.fetchStartWed(1101)).thenReturn(defWedStart);  when(dbCon.fetchEndWed(1101)).thenReturn(defWedEnd);  when(dbCon.fetchStartThu(1101)).thenReturn(defThuStart);  when(dbCon.fetchEndThu(1101)).thenReturn(defThuEnd);  when(dbCon.fetchStartFri(1101)).thenReturn(defFriStart);  when(dbCon.fetchEndFri(1101)).thenReturn(defFriEnd);  when(dbCon.fetchStartSat(1101)).thenReturn(defSatStart);  when(dbCon.fetchEndSat(1101)).thenReturn(defSatEnd);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.getDataValues() |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getData\_Rainy\_006 |
| Test Purpose: | Test if the getData method retrieves data from database and stores it in variables |
| Test Setup | defSub = "";  defSemester = "Spring";  defCourseName = "English";  defCourseStart = "01/08/19";  defCourseEnd = "05/01/19";  defMonStart = "15:00";  defMonEnd = "16:00";  defTueStart = "15:00";  defTueEnd = "16:00";  defWedStart = "15:00";  defWedEnd = "16:00";  defThuStart = "15:00";  defThuEnd = "16:00";  defFriStart = "15:00";  defFriEnd = "16:00";  defSatStart = "15:00";  defSatEnd = "16:00";  String[] allValues = {defSub, defSemester, defCourseName, defCourseStart,defCourseEnd, defMonStart, defMonEnd, defTueStart,  defTueEnd, defWedStart, defWedEnd, defThuStart, defThuEnd, defFriStart, defFriEnd, defSatStart, defSatEn };  when(dbCon.fetchCourseSubj(1101)).thenReturn(defSub);  when(dbCon.fetchCourseSemester(1101)).thenReturn(defSemester);  when(dbCon.fetchCourseName(1101)).thenReturn(defCourseName);  when(dbCon.fetchCourseStart(1101)).thenReturn(defCourseStart);  when(dbCon.fetchCourseEnd(1101)).thenReturn(defCourseEnd);  when(dbCon.fetchStartMon(1101)).thenReturn(defMonStart);  when(dbCon.fetchEndMon(1101)).thenReturn(defMonEnd);  when(dbCon.fetchStartTue(1101)).thenReturn(defTueStart);  when(dbCon.fetchEndTue(1101)).thenReturn(defTueEnd);  when(dbCon.fetchStartWed(1101)).thenReturn(defWedStart);  when(dbCon.fetchEndWed(1101)).thenReturn(defWedEnd);  when(dbCon.fetchStartThu(1101)).thenReturn(defThuStart);  when(dbCon.fetchEndThu(1101)).thenReturn(defThuEnd);  when(dbCon.fetchStartFri(1101)).thenReturn(defFriStart);  when(dbCon.fetchEndFri(1101)).thenReturn(defFriEnd);  when(dbCon.fetchStartSat(1101)).thenReturn(defSatStart);  when(dbCon.fetchEndSat(1101)).thenReturn(defSatEnd);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.getDataValues |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_LogIn\_Sunny\_007 |
| Test Purpose: | Test if the LogIn method successfully logs into the database with specific username/password |
| Test Setup: | when(dbCon.connect("user1", "1234")).thenReturn(0);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.LogIn();  controller.getLoggedIn() |
| Expected Output: | States changed:  loggedIn = true |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_LogIn\_Rainy\_008 |
| Test Purpose: | Test if the LogIn method successfully logs into the database with specific username/password |
| Test Setup: | when(dbCon.connect("user1", "1234")).thenReturn(-1);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.LogIn();  controller.getLoggedIn() |
| Expected Output: | Stateschanged  loggedIn = false |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getCon\_Sunny\_009 |
| Test Purpose: | Test if the getCon method successfully retrieves the database instance from class |
| Test Setup: | See Table 3.1 for setUp mocks. |
| Test Input: | controller.getCon() |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getCon\_Rainy\_010 |
| Test Purpose: | Test if the getCon method successfully retrieves the database instance from class |
| Test Setup: | DBConnection otherdbCon = mock(DBConnection.class);  See Table 3.1 for setUp mocks. |
| Test Input: | controller. getCon() |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_timerParser\_Sunny\_011 |
| Test Purpose: | Test if the timerParser method correctly gets a time string from database and parses it into int variables of class |
| Test Setup: | time = “10:30”  See Table 3.1 for setUp mocks. |
| Test Input: | controller.timeParser(time); |
| Expected Output: | States:  Hr = 10  Min = 30 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_timerParser\_Rainy\_012 |
| Test Purpose: | Test if the timerParser method correctly gets a time string from database and parses it into int variables of class |
| Test Setup: | time = “11:50”  See Table 3.1 for setUp mocks. |
| Test Input: | controller.timeParser(time); |
| Expected Output: | States:  Hr = 10  Min = 30 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_dateParser\_Sunny\_013 |
| Test Purpose: | Test if the dateParser method correctly gets a date string from database and parses it into int variables of class |
| Test Setup: | date = “01/01/19”  See Table 3.1 for setUp mocks. |
| Test Input: | controller.timeParser(date); |
| Expected Output: | States:  ClearMonth = 01  ClearDate = 01  Clearyear = 19 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_dateParser\_Rainy\_014 |
| Test Purpose: | Test if the dateParser method correctly gets a date string from database and parses it into int variables of class |
| Test Setup: | Date = “02/02/18”  See Table 3.1 for setUp mocks. |
| Test Input: | controller.dateParser(date); |
| Expected Output: | States:  ClearMonth = 01  ClearDate = 01  Clearyear = 19 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_ReturnHr\_Sunny\_015 |
| Test Purpose: | Test if the ReturnHr method correctly retrieves the hr variable from class |
| Test Setup: | Hr = 0  See Table 3.1 for setUp mocks. |
| Test Input: | controller.returnHr(); |
| Expected Output: | 0 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_ReturnHr\_Rainy\_016 |
| Test Purpose: | Test if the ReturnHr method correctly retrieves the hr variable from class |
| Test Setup: | Hr = 0  See Table 3.1 for setUp mocks. |
| Test Input: | controller.returnHr(); |
| Expected Output: | 1 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_ReturnMin\_Sunny\_017 |
| Test Purpose: | Test if the ReturnMin method correctly retrieves the min variable from class |
| Test Setup: | Min = 0  See Table 3.1 for setUp mocks. |
| Test Input: | controller.returnMin(); |
| Expected Output: | 0 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_ReturnMin\_Rainy\_018 |
| Test Purpose: | Test if the ReturnMin method correctly retrieves the min variable from class |
| Test Setup: | Min = 0  See Table 3.1 for setUp mocks. |
| Test Input: | controller.returnMin(); |
| Expected Output: | 1 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getEndTime\_Sunny\_019 |
| Test Purpose: | Test if the getEndTime method correctly checks current time against the scheduled end time of class |
| Test Setup: | Calender newC = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30 - 1, 1);  date = newC.getTime();    See Table 3.1 for setUp mocks. |
| Test Input: | time1 = 10  time2 = 30  controller.getEndTime(time1, time2); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getEndTime\_Rainy\_020 |
| Test Purpose: | Test if the getEndTime method correctly checks current time against the scheduled end time of class |
| Test Setup: | Calender newC = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30, 1);  date = newC.getTime();    See Table 3.1 for setUp mocks. |
| Test Input: | time1 = 10  time2 = 30  controller.getEndTime(time1, time2); |
| Expected Output: | False |
| Test ID: | Unit\_Test\_getSemesterClear\_Sunny\_021 |
| Test Purpose: | Test if the getSemesterClear method correctly gets the new date to clear semester from class |
| Test Setup: | Calender autoClear = new GregorianCalendar();  autoClear.set(19, 01, 01, 10, 30);  date = autoClear.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | year = 19  month = 01  day = 01  hr = 10  min = 30  controller.setSemesterClear(year, month, day, hr, min); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getSemesterClear\_Rainy\_022 |
| Test Purpose: | Test if the getSemesterClear method correctly gets the new date to clear semester from class |
| Test Setup: | Calender autoClear = new GregorianCalendar();  autoClear.set(19, 01, 01, 10, 30);  date = autoClear.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | year = 20  month = 02  day = 02  hr = 12  min = 40  controller.setSemesterClear(year, month, day, hr, min); |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getSemesterClear\_Sunny\_023 |
| Test Purpose: | Test if the getSemesterClear method correctly gets the new date to clear semester from class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  Set controller.autoClear = current date  See Table 3.1 for setUp mocks. |
| Test Input: | autoClear1.getTime().getTime() |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getSemesterClear\_Rainy\_024 |
| Test Purpose: | Test if the getSemesterClear method correctly gets the new date to clear semester from class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  autoClear1.set(2019, 01,01,11,55)  Set controller.autoClear = current date  See Table 3.1 for setUp mocks. |
| Test Input: | autoClear1.getTime().getTime() |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_get15BeforeEnd\_Sunny\_025 |
| Test Purpose: | Test if the get15BeforeEnd method correctly gets the time 15 mins before the end of schedule class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30 - 15, 1);  date = newC.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | hr = 10  min = 30  controller.get15BeforeEnd(hr, min) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_get15BeforeEnd\_Rainy\_026 |
| Test Purpose: | Test if the get15BeforeEnd method correctly gets the time 15 mins before the end of schedule class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30, 1);  date = newC.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | hr = 10  min = 30  controller.get15BeforeEnd(hr, min) |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_get5BeforeEnd\_Sunny\_027 |
| Test Purpose: | Test if the get 5BeforeEnd method correctly gets the time 5 mins before the end of schedule class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30 - 5, 1);  date = newC.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | hr = 10  min = 30  controller.get5BeforeEnd(hr, min) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_get5BeforeEnd\_Rainy\_028 |
| Test Purpose: | Test if the get 5BeforeEnd method correctly gets the time 5 mins before the end of schedule class |
| Test Setup: | Calender autoClear1 = new GregorianCalendar();  years = newC.get(newC.YEAR);  months = newC.get(newC.MONTH);  dates = newC.get(newC.DATE);  newC.set(years, months, dates, 10, 30, 1);  date = newC.getTime();  See Table 3.1 for setUp mocks. |
| Test Input: | hr = 10  min = 30  controller.get5BeforeEnd(hr, min) |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_test\_loggedin\_Sunny\_029 |
| Test Purpose: | Test if the loggedin method, containing the login logic for main, correctly performs login operations |
| Test Setup: | Do nothing when form is called  doNothing().when(icMock).Initiate\_Login\_Form();  mock log class of interfaceController mock  icMock.log = logMock;  when(icMock.log.dataReceived()).thenReturn(true);  when(icMock.log.getUsername()).thenReturn("user1");  when(icMock.log.getPassword()).thenReturn("1234");  when(authMock.validate\_Login()).thenReturn(true);  when(dbCon.connect("user1", "1234")).thenReturn(0);  See Table 3.1 for setUp mocks. |
| Test Input: | controller.loggedin() |
| Expected Output: | States  loggedin = true |

|  |  |
| --- | --- |
| Test ID: | Unit\_test\_loggedin\_Rainy\_030 |
| Test Purpose: | Test if the loggedin method, containing the login logic for main, correctly performs login operations |
| Test Setup: | Do nothing when form is called  doNothing().when(icMock).Initiate\_Login\_Form();  set log and msg class of interfaceController mock  icMock.log = logMock;  icMock.msg = msgMock  when(icMock.log.dataReceived()).thenReturn(true);  when(icMock.log.getUsername()).thenReturn("user1");  when(icMock.log.getPassword()).thenReturn("1234");  when(authMock.validate\_Login()).thenReturn(false);  when(dbCon.connect("user1", "1234")).thenReturn(0);  doNothing().when(icMock).Initiate\_IncorrectLogin();  icMock.msg.ack = true;  See Table 3.1 for setUp mocks. |
| Test Input: | controller.loggedin() |
| Expected Output: | States  loggedin = true |

**DBConnect Test Cases**

|  |  |
| --- | --- |
| @Before  setUp | myDB = new DbConnection() |
| @After  tearDown | myDB = null |

Table 3.2 Mock setUp and tearDown called before and after mocks respectively.

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_connect\_Sunny\_031 |
| Test Purpose: | Test if the connect method correctly stablishes a connection with the database |
| Test Setup: | myCon = mock(Connection.class) |
| Test Input: | myDB.connect(“user1”, “1234”, myCon); |
| Expected Output: | 0 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_connect\_Rainy\_032 |
| Test Purpose: | Test if the connect method correctly stablishes a connection with the database |
| Test Setup: | myCon = null; |
| Test Input: | myDB.connect(“usdsds”, “1234”, myCon); |
| Expected Output: | -1 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_disconnect\_Sunny\_033 |
| Test Purpose: | Test if the disconnect method correctly disconnects from the database |
| Test Setup: | myCon = mock(Connection.class)  myDB.setCon(myCon); |
| Test Input: | myDB.disconnect(); |
| Expected Output: | 0 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_disconnect\_Rainy\_034 |
| Test Purpose: | Test if the disconnect method correctly disconnects from the database |
| Test Setup: | myDB.setCon(null); |
| Test Input: | myDB.disconnect(); |
| Expected Output: | -1 |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseID\_Sunny\_035 |
| Test Purpose: | Test if the fetchCourseID method gets the course id from the database given a specific course id parameter |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_id”)).thenReturn(4555);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseID(4555); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseID\_Rainy\_036 |
| Test Purpose: | Test if the fetchCourseID method gets the course id from the database given a specific course id parameter |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_id”)).thenReturn(3444);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseID(3444); |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getEndDates\_Sunny\_037 |
| Test Purpose: | Test if the getEndDates method gets the end gates of a semester from the database given a specific course id |
| Test Setup: | Arraylist<String> state = new ArrayList<String>();  State.add(“05/01/19”);  State.add(“05/01/19”);  State.add(“05/01/19”);  Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getString(“end\_date”)).thenReturn(“05/01/19”, “05/01/19”, “05/01/19”);  myDB.setCon(myCon); |
| Test Input: | myDb.getEndDates(); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getEndDates\_Rainy\_038 |
| Test Purpose: | Test if the getEndDates method gets the end gates of a semester from the database given a specific course id |
| Test Setup: | Arraylist<String> state = new ArrayList<String>();  State.add(“05/01/19”);  State.add(“05/01/19”);  State.add(“05/01/19”);  Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getString(“end\_date”)).thenReturn(“05/06/19”, “05/02/19”, “05/20/19”);  myDB.setCon(myCon); |
| Test Input: | myDb.getEndDates(); |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getCourses\_Sunny\_039 |
| Test Purpose: | Test if the getCourses method returns a list with all the course id of all the courses in the database |
| Test Setup: | Arraylist<Integer> state = new ArrayList<Integer>();  State.add(1101);  State.add(4555);  State.add(4610);  Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.next()).thenReturn(true, true, true, false);  when(result.getInt(“course\_id”)).thenReturn(1101, 4555, 4610);  myDB.setCon(myCon); |
| Test Input: | myDb.getCourses() |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_getCourses\_Rainy\_040 |
| Test Purpose: | Test if the getCourses method returns a list with all the course id of all the courses in the database |
| Test Setup: | Arraylist<Integer> state = new ArrayList<Integer>();  State.add(1101);  State.add(4555);  State.add(4610);  Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.next()).thenReturn(true, true, true, false);  when(result.getInt(“course\_id”)).thenReturn(1100, 4500, 4600);  myDB.setCon(myCon); |
| Test Input: | myDb.getCourses() |
| Expected Output: | False |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseSubj\_Sunny\_041 |
| Test Purpose: | Test if the fetchCourseSubj method returns the course subject from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_subject”)).thenReturn(“English”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseSubj(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseSubj\_Rainy\_042 |
| Test Purpose: | Test if the fetchCourseSubj method returns the course subject from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_subject”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseSubj(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseName\_Sunny\_043 |
| Test Purpose: | Test if the fetchCourseName method returns the course name from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_name”)).thenReturn(“ENC”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseName(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseName\_Rainy\_044 |
| Test Purpose: | Test if the fetchCourseName method returns the course name from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“course\_name”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseName(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseSemester\_Sunny\_045 |
| Test Purpose: | Test if the fetchCourseSemester method returns the semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“semester”)).thenReturn(“Spring”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseSemester(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseSemester\_Rainy\_046 |
| Test Purpose: | Test if the fetchCourseSemester method returns the semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“semester”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseSemester(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseStart\_Sunny\_047 |
| Test Purpose: | Test if the fetchCourseStart method returns the start date of semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_date”)).thenReturn(“01/08/19”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseStart(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseStart\_Rainy\_048 |
| Test Purpose: | Test if the fetchCourseStart method returns the start date of semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_date”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseStart(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseEnd\_Sunny\_049 |
| Test Purpose: | Test if the fetchCourseEnd method returns the end date of semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_date”)).thenReturn(“05/01/19”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseStart(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchCourseEnd\_Rainy\_050 |
| Test Purpose: | Test if the fetchCourseEnd method returns the end date of semester from the database given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_date”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchCourseStart(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartMon\_Sunny\_051 |
| Test Purpose: | Test if the fetchStartMon method correctly returns the time a course starts on Monday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_mon”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartMon(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartMon\_Rainy\_052 |
| Test Purpose: | Test if the fetchStartMon method correctly returns the time a course starts on Monday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_mon”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartMon(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndMon\_Sunny\_053 |
| Test Purpose: | Test if the fetchEndtMon method correctly returns the time a course ends on Monday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_mon”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndMon(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndMon\_Rainy\_054 |
| Test Purpose: | Test if the fetchEndtMon method correctly returns the time a course ends on Monday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_mon”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndMon(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartTue\_Sunny\_055 |
| Test Purpose: | Test if the fetchStartTue method correctly returns the time a course starts on Tuesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_tue”)).thenReturn(“15:00”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartTue (1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartTue\_Rainy\_056 |
| Test Purpose: | Test if the fetchStartTue method correctly returns the time a course starts on Tuesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_tue”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartTue(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndTue\_Sunny\_057 |
| Test Purpose: | Test if the fetchEndTue method correctly returns the time a course ends on Tuesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_tue”)).thenReturn(“16:55”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndTue(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndTue\_Rainy\_058 |
| Test Purpose: | Test if the fetchEndTue method correctly returns the time a course ends on Tuesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_tue”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndTue(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartWed\_Sunny\_059 |
| Test Purpose: | Test if the fetchStartWed method correctly returns the time a course starts on Wednesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_wed”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartWed(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartWed\_Rainy\_060 |
| Test Purpose: | Test if the fetchStartWed method correctly returns the time a course starts on Wednesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_wed”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartWed(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndWed\_Sunny\_061 |
| Test Purpose: | Test if the fetchEndWed method correctly returns the time a course ends on Wednesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_wed”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndWed(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndWed\_Rainy\_062 |
| Test Purpose: | Test if the fetchEndWed method correctly returns the time a course ends on Wednesday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_wed”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndWed(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartThu\_Sunny\_063 |
| Test Purpose: | Test if the fetchStartThu method correctly returns the time a course starts on Thursday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_thu”)).thenReturn(“15:00”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartThu(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStarThu\_Rainy\_064 |
| Test Purpose: | Test if the fetchStartThu method correctly returns the time a course starts on Thursday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_thu”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartThu(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndThu\_Sunny\_065 |
| Test Purpose: | Test if the fetchEndThu method correctly returns the time a course ends on Thursday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_thu”)).thenReturn(“16:55”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndThu(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndThu\_Rainy\_066 |
| Test Purpose: | Test if the fetchEndThu method correctly returns the time a course ends on Thursday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_thu”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndThu(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartFri\_Sunny\_067 |
| Test Purpose: | Test if the fetchStartFri method correctly returns the time a course starts on Friday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_fri”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartFri(1101); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartFri\_Rainy\_068 |
| Test Purpose: | Test if the fetchStartFri method correctly returns the time a course starts on Friday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_fri”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartFri(1100); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndFri\_Sunny\_069 |
| Test Purpose: | Test if the fetchEndFri method correctly returns the time a course ends on Friday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_fri”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndFri(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndFri\_Rainy\_070 |
| Test Purpose: | Test if the fetchEndFri method correctly returns the time a course ends on Friday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_fri”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndFri(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartSat\_Sunny\_071 |
| Test Purpose: | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_sat”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchStartSat(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchStartSat\_Rainy\_072 |
| Test Purpose: | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“start\_sat”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndFri(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndSat\_Sunny\_073 |
| Test Purpose: | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_sat”)).thenReturn(null);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndSat(1101) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_fetchEndSat\_Rainy\_074 |
| Test Purpose: | Test if the fetchStartSat method correctly returns the time a course starts on Saturday given a specific course id |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  result = mock(ResultSet.class);  when(myCon.createStatement()).thenReturn(statement);  when(statement.getResultSet()).thenReturn(result);  when(result.getInt(“end\_sat”)).thenReturn(“Not Found”);  myDB.setCon(myCon); |
| Test Input: | myDb.fetchEndSat(1100) |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_storeClassInfo\_Sunny\_075 |
| Test Purpose: | Test if the storeClassInfo method correctly stores the class info into database. |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  when(myCon.createStatement()).thenReturn(statement);  myDB.setCon(myCon); |
| Test Input: | myDb.storeClassInfo(1101, “ENC”, “English”, Spring”); |
| Expected Output: | True |

|  |  |
| --- | --- |
| Test ID: | Unit\_Test\_storeClassInfo\_Rainy\_076 |
| Test Purpose: | Test if the storeClassInfo method correctly stores the class info into database. |
| Test Setup: | Statement = mock(Statement.class);  myCon = mock(Connection.class);  when(myCon.createStatement()).thenReturn(statement);  myDB.setCon(myCon); |
| Test Input: | myDb.storeClassInfo(1101, “ENC”, “English”, Spring”); |
| Expected Output: | False |

**3.2 Actual Test Results**

|  |  |
| --- | --- |
| Test ID | Result |
| Unit\_Test\_checkClear\_Sunny\_001 | Pass |
| Unit\_Test\_checkClear\_Rainy\_002 | Pass |
| Unit\_Test\_checkTimes\_Sunny\_003 | Pass |
| Unit\_Test\_checkTimes\_Rainy\_004 | Pass |
| Unit\_Test\_getData\_Sunny\_005 | Pass |
| Unit\_Test\_getData\_Rainy\_006 | Pass |
| Unit\_Test\_LogIn\_Sunny\_007 | Pass |
| Unit\_Test\_LogIn\_Rainy\_008 | Pass |
| Unit\_Test\_getCon\_Sunny\_009 | Pass |
| Unit\_Test\_getCon\_Rainy\_010 | Pass |
| Unit\_Test\_timerParser\_Sunny\_011 | Pass |
| Unit\_Test\_timerParser\_Rainy\_012 | Pass |
| Unit\_Test\_dateParser\_Sunny\_013 | Pass |
| Unit\_Test\_dateParser\_Rainy\_014 | Pass |
| Unit\_Test\_ReturnHr\_Sunny\_015 | Pass |
| Unit\_Test\_ReturnHr\_Rainy\_016 | Pass |
| Unit\_Test\_ReturnMin\_Sunny\_017 | Pass |
| Unit\_Test\_ReturnMin\_Rainy\_018 | Pass |
| Unit\_Test\_getEndTime\_Sunny\_019 | Pass |
| Unit\_Test\_getEndTime\_Rainy\_020 | Pass |
| Unit\_Test\_getSemesterClear\_Sunny\_021 | Pass |
| Unit\_Test\_getSemesterClear\_Rainy\_022 | Pass |
| Unit\_Test\_setSemesterClear\_Sunny\_023 | Pass |
| Unit\_Test\_setSemesterClear\_Rainy\_024 | Pass |
| Unit\_Test\_get15BeforeEnd\_Sunny\_025 | Pass |
| Unit\_Test\_get15BeforeEnd\_Rainy\_026 | Pass |
| Unit\_Test\_get5BeforeEnd\_Sunny\_027 | Pass |
| Unit\_Test\_get5BeforeEnd\_Rainy\_028 | Pass |
| Unit\_test\_loggedin\_Sunny\_029 | Pass |
| Unit\_test\_loggedin\_Rainy\_030 | Fail |
| Test ID | Result |
| Unit\_Test\_connect\_Sunny\_031 | Pass |
| Unit\_Test\_connect\_Rainy\_032 | Fail |
| Unit\_Test\_disconnect\_Sunny\_033 | Pass |
| Unit\_Test\_disconnect\_Rainy\_034 | Fail |
| Unit\_Test\_fetchCourseID\_Sunny\_035 | Pass |
| Unit\_Test\_fetchCourseID\_Rainy\_036 | Pass |
| Unit\_Test\_getEndDates\_Sunny\_037 | Pass |
| Unit\_Test\_getEndDates\_Rainy\_038 | Pass |
| Unit\_Test\_getCourses\_Sunny\_039 | Pass |
| Unit\_Test\_getCourses\_Rainy\_040 | Pass |
| Unit\_Test\_fetchCourseSubj\_Sunny\_041 | Pass |
| Unit\_Test\_fetchCourseSubj\_Rainy\_042 | Pass |
| Unit\_Test\_fetchCourseName\_Sunny\_043 | Pass |
| Unit\_Test\_fetchCourseName\_Rainy\_044 | Pass |
| Unit\_Test\_fetchCourseSemester\_Sunny\_045 | Pass |
| Unit\_Test\_fetchCourseSemester\_Rainy\_046 | Pass |
| Unit\_Test\_fetchCourseStart\_Sunny\_047 | Pass |
| Unit\_Test\_fetchCourseStart\_Rainy\_048 | Pass |
| Unit\_Test\_fetchCourseEnd\_Sunny\_049 | Pass |
| Unit\_Test\_fetchCourseEndt\_Rainy\_050 | Pass |
| Unit\_Test\_fetchStartMon\_Sunny\_051 | Pass |
| Unit\_Test\_fetchStartMon\_Rainy\_052 | Pass |
| Unit\_Test\_fetchEndMon\_Sunny\_053 | Pass |
| Unit\_Test\_fetchEndMon\_Rainy\_054 | Pass |
| Unit\_Test\_fetchStartTue\_Sunny\_055 | Pass |
| Unit\_Test\_fetchStartTue\_Rainy\_056 | Pass |
| Unit\_Test\_fetchEndTue\_Sunny\_057 | Pass |
| Unit\_Test\_fetchEndTue\_Rainy\_058 | Pass |
| Unit\_Test\_fetchStartWed\_Sunny\_059 | Pass |
| Unit\_Test\_fetchStartWed\_Rainy\_060 | Pass |
| Unit\_Test\_fetchEndWed\_Sunny\_061 | Pass |
| Unit\_Test\_fetchEndWed\_Rainy\_062 | Pass |
| Unit\_Test\_fetchStartThu\_Sunny\_063 | Pass |
| Unit\_Test\_fetchStartThu\_Rainy\_064 | Pass |
| Unit\_Test\_fetchEndThu\_Sunny\_065 | Pass |
| Unit\_Test\_fetchEndThu\_Rainy\_066 | Pass |
| Unit\_Test\_fetchStartFri\_Sunny\_067 | Pass |
| Unit\_Test\_fetchStartFri\_Rainy\_068 | Pass |
| Unit\_Test\_fetchEndFri\_Sunny\_069 | Pass |
| Unit\_Test\_fetchEndFri\_Rainy\_070 | Pass |
| Unit\_Test\_fetchStartSat\_Sunny\_071 | Pass |
| Unit\_Test\_fetchStartSat\_Rainy\_072 | Pass |
| Unit\_Test\_fetchEndSat\_Sunny\_073 | Pass |
| Unit\_Test\_fetchEndSat\_Rainy\_074 | Pass |
| Unit\_Test\_storeClassInfo\_Sunny\_075 | Pass |
| Unit\_Test\_storeClassInfo\_Rainy\_076 | Fail |

**4. Subsystem Testing**

This chapter focuses on subsystem testing of PSM\_Logic package. These includes test cases, procedures, and test result of the logic package.

**4.1 Subsystem Test Cases**

There is a total of six test cases for subsystem testing. These tests are to determine if PSM\_Logic package is able to correctly retrieve information from the database and return it to the client. This section contains the test cases, their objectives, criteria, and the procedures for testing.

**4.1.1 Test Identification and Objective**

The test cases below shows the test ID and objectives of each case performed for the PSM\_Logic package. The values for the test stubs where either hard coded in the returns or mocked.

|  |  |
| --- | --- |
| Test ID | Purpose |
| courseSelectTest\_Subsys\_Sunny001 | This test verifies if the appController is able to retrieve a hard-coded string from the DBConnection fetchCourse method. |
| courseSelectTest\_Subsys\_Sunny002 | This is another simple test case that returns a hard-coded string. |
| LogicFacadeTest\_Subsys\_Sunny003 | This test verifies that getData returns the data for a course. |
| courseSelectTest\_Subsys\_Rainy004 | This test verifies that if there are no courses available that it will return null. |
| LogicFacadeTest\_Subsys\_Rainy005 | Verify that getData is able to return a null when there is no course info available. |
| courseSelectTest\_Subsys\_Rainy006 | This is another simple test to verify that courseSelect gives an error when the string values do not match. |

**4.1.2 Test Criteria and Procedures**

Test Criteria

The logic package was removed to see where the dependency of appController was. In the interface package courseSelect has that dependency so the test was conducted using equivalence partition with illegal and legal values. First a string was coded to the return value of the method in DBConnection called fetchCourses, then a Junit test case was conducted to see if the test would return a legal value which means the test passed and illegal value which means the test failed.

Procedures

When the methods were called a return type was hard coded into the method to see if the Logic package was able to return a valid answer. Junit will create the appropriate test stubs where we will then assert the expected value to be acquired from that method call. When the test gets a pass or fail it is then documented and another test is conducted.

**4.1.3 Test Cases**

|  |  |
| --- | --- |
| Test ID: | courseSelectTest\_Subsys\_Sunny001 |
| Test Purpose: | This is a simple test to verify that courseSelect is able to successfully fetch a string that was hard coded into the return of the fecthCourses method of DBConnection. The appController is the class in the logic package that is responsible for retrieving it from DBConnection. |
| Test Setup: | In this setup the return value was hard coded. |
| Test Input: | No input retrieving data from DBConnection. |
| Expected Output: | Course Schedule: CEN4072, CEN4010, COP4520, CAP4770 |

|  |  |
| --- | --- |
| Test ID: | courseSelectTest\_Subsys\_Sunny002 |
| Test Purpose: | This is another simple test to verify that courseSelect is able to successfully fetch a string that was hard coded into the return of the fecthCourses method of DBConnection. The appController is the class in the logic package that is responsible for retrieving it from DBConnection. |
| Test Setup: | In this setup the return value was hard coded. |
| Test Input: | No input retrieving data from DBConnection. |
| Expected Output: | Course Schedule: ENC1101, MAC1147, STA3033, PHY1049 |

|  |  |
| --- | --- |
| Test ID: | LogicFacadeTest\_Subsys\_Sunny003 |
| Test Purpose: | This test was made to see if appController method is able to return an array string from all the methods called in LogicFacade. |
| Test Setup: | In this setup the return value was hard coded. |
| Test Input: | 1101 |
| Expected Output: | An array with this information {"ENC", "Spring", "English", "01/08/19", "05/01/19",  "15:00", "16:00", "15:00", "16:00", "15:00", "16:00", "15:00", "16:00", "15:00", "16:00", "15:00", "16:00"} |

|  |  |
| --- | --- |
| Test ID: | courseSelectTest\_Subsys\_Rainy004 |
| Test Purpose: | Verify that getData is able to return a null when there is no course info available. |
| Test Setup: | In this setup the return value was hard coded. |
| Test Input: | No input retrieving data from DBConnection. |
| Expected Output: | null |

|  |  |
| --- | --- |
| Test ID: | LogicFacadeTest\_Subsys\_Rainy005 |
| Test Purpose: | Verify that courseSelect is able to return null when there are no courses to return. |
| Test Setup: | In this setup nothing was mocked or changed, and an input was given. |
| Test Input: | No input retrieving data from DBConnection. |
| Expected Output: | null |

|  |  |
| --- | --- |
| Test ID: | courseSelectTest\_Subsys\_Rainy006 |
| Test Purpose: | This is another simple test to verify that courseSelect will give an error when the string values do not match. |
| Test Setup: | In this setup the return value was hard coded. |
| Test Input: | No input retrieving data from DBConnection. |
| Expected Output: | A failure |

**4.2 Actual Test Results**

|  |  |
| --- | --- |
| Test ID | Result |
| courseSelectTest\_Subsys\_Sunny001 | Pass |
| courseSelectTest\_Subsys\_Sunny002 | Pass |
| LogicFacadeTest\_Subsys\_Sunny003 | Fail |
| courseSelectTest\_Subsys\_Rainy001 | Fail |
| LogicFacadeTest\_Subsys\_Rainy002 | Fail |
| courseSelectTest\_Subsys\_Rainy003 | Fail |

**5. System Testing**

**5.1 System Test Cases**

There are a total of 22 System test cases covering the implemented use cases: Login, add class schedule, edit class schedule and password conflicts. For each user case covered, there are 3 sunny day and 3 rainy day test cases.

**5.1.1 Test Identification and Objective**

|  |  |  |
| --- | --- | --- |
| Case ID | | Purpose |
| System\_Login\_Sunny01 | | To verify that a registered user can login to the PMS. |
| System\_Login\_Sunny02 | |
| System\_Login\_Sunny03 | |
| System\_Login\_Rainy04 | | To verify that an unregistered user cannot login to the system. |
| System\_Login\_Rainy05 | |
| System\_Login\_Rainy06 | |
| System\_AddClass\_Sunny07 | | To verify that the user can properly register classes to the database |
| System\_AddClass\_Sunny08 | |
| System\_AddClass\_Sunny09 | |
| System\_AddClass\_Rainy10 | | To verify that the user can’t register a class with an incorrect value to the database. |
| System\_AddClass\_Rainy11 | |
| System\_AddClass\_Rainy12 | |
| System\_EditSchedule\_Sunny13 | | To verify that a professor can update his schedule |
| System\_EditSchedule\_Sunny14 | |
| System\_EditSchedule\_Sunny15 | |
| System\_EditSchedule\_Rainy16 | | To verify that an incorrect class ID or an employee without the proper credentials can’t update the database. |
| System\_EditSchedule\_Rainy17 | |
| System\_EditSchedule\_Rainy18 | |
| System\_PasswordConflicts\_Sunny19 | | To verify that the app closes if enough failed logins are attempted. |
| System\_PasswordConflicts\_Sunny20 | |
| System\_PasswordConflicts\_Sunny21 | |
| System\_PasswordConflicts\_Rainy22 | To verify the app closes if enough failed logins are attempted | |
| System\_PasswordConflicts\_Rainy23 |
| System\_PasswordConflicts\_Rainy24 |

**5.1.2 Test Criteria and Procedures**

Test Criteria

Each of the implemented test cases result in the display of a new window, the change of data in the database, display of an error message, or to no output. Therefore, valid inputs will result in the display of a new window or that alter the information in the database. Invalid inputs will result in the display of an error message or to no output.

Test Procedure

A MySQL server running on localhost:3306 is used to store the schedule. The system tests are performed using Rational Functional Manager v9.2.1. The corresponding SQL connector libraries were added to RFT in order to reinitialize the database before tests. Sunny days and rainy days scripts and datapools are used to perform the tests.

**5.1.3 Test Cases**

|  |  |
| --- | --- |
| Test ID | System\_Login\_Sunny01 |
| Test Purpose | To test that a registered user can properly login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “pclarke”  Password: “1234” |
| Expected output | The app opens PSM Main Menu |

|  |  |
| --- | --- |
| Test ID | System\_Login\_Sunny02 |
| Test Purpose | To test that a registered user can properly login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “user1”  Password: “1234” |
| Expected output | The app opens PSM Main Menu |

|  |  |
| --- | --- |
| Test ID | System\_Login\_Sunny03 |
| Test Purpose | To test that a registered user can properly login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “user2”  Password: “5678” |
| Expected output | The app opens PSM Main Menu |

|  |  |
| --- | --- |
| Test ID | System\_Login\_Rainy04 |
| Test Purpose | To test that an unregistered user can’t login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “pclarke”  Password: “wrongpassword” |
| Expected output | The app displays “Incorrect Username/Password” |

|  |  |
| --- | --- |
| Test ID | System\_Login\_Rainy05 |
| Test Purpose | To test that an unregistered user can’t login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “wronguser”  Password: “1234” |
| Expected output | The app displays “Incorrect Username/Password” |

|  |  |
| --- | --- |
| Test ID | System\_Login\_Rainy06 |
| Test Purpose | To test that an unregistered user can’t login. |
| Test Setup | Reinitialize the database to its initial state |
| Test Input | Username: “wronguser”  Password: “wrongpassword” |
| Expected output | The app displays “Incorrect Username/Password” |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Sunny07 |
| Test Purpose | To verify that the user can properly register classes to the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Add Class Schedule”  Semester: “Spring”  Start Date = “01/07/19”  End Date = “04/25/19”  Subject = “COP”  Course Number = “4710”  Course Name = “Database”  Start time Monday = “1825”  End Time Monday = “1940”  Start Time Wednesday = “1825”  End Time Wednesday = “1940” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Sunny08 |
| Test Purpose | To verify that the user can properly register classes to the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Add Class Schedule”  Semester: “Spring”  Start Date = “01/11/19”  End Date = “04/29/19”  Subject = “CEN”  Course Number = “4072”  Course Name = “Software Testing”  Start time Tuesday = “1500”  End Time Tuesday = “1700”  Start Time Thursday = “1500”  End Time Thursday = “1700” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Sunny09 |
| Test Purpose | To verify that the user can properly register classes to the database |
| Test Setup | Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Add Class Schedule”  Semester: “Spring”  Start Date = “08/12/19”  End Date = “12/10/19”  Subject = “CHM”  Course Number = “1045”  Course Name = “Chemistry 1”  Start time Monday = “1225”  End Time Monday = “1440”  Start Time Wednesday = “1225”  End Time Wednesday = “1440” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Rainy10 |
| Test Purpose | To verify that the user can’t register a class with an incorrect value to the database. |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234”   Click “Add Class Schedule” |
| Test Input | Semester: “Spring”  Start Date = “08/12/19”  End Date = “12/10/19”  Subject = “CHM”  Course Number = “AAA”  Course Name = “Chemistry 1”  Start time Tuesday = “1200”  End Time Tuesday = “1400”  Start Time Thursday = “1200”  End Time Thursday = “1400” |
| Expected output | No expected output |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Rainy11 |
| Test Purpose | To verify that the user can’t register a class with an incorrect value to the database. |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234”   Click “Add Class Schedule” |
| Test Input | Semester: “Summer A”  Start Date = “05/10/19”  End Date = “07/27/19”  Subject = “COP”  Course Number = “-4555”  Course Name = “Fundamentals of Programming Languages”  Start time Tuesday = “1200”  End Time Tuesday = “1500”  Start Time Thursday = “1200”  End Time Thursday = “1500” |
| Expected output | No expected output |

|  |  |
| --- | --- |
| Test ID | System\_AddClass\_Rainy12 |
| Test Purpose | To verify that the user can’t register a class with an incorrect value to the database. |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234”   Click “Add Class Schedule” |
| Test Input | Semester: “Spring”  Start Date = “800/12/2019”  End Date = “12/10/2019”  Subject = “ENC”  Course Number = “1012”  Course Name = “Writing 2”  Start time Tuesday = “1200”  End Time Tuesday = “1400”  Start Time Thursday = “1200”  End Time Thursday = “1400” |
| Expected output | No expected output |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Sunny13 |
| Test Purpose | To verify that the user can properly register changes to their classes to the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select: 4555  Start time Monday = “1200”  End Time Monday = “1400”  Start Time Wednesday = “1200”  End Time Wednesday = “1400”  Start Time Friday = “1200”  End Time Friday = “1400”  Start Time Tuesday =””  End Time Tuesday = “”  Start Time Thursday =””  End Time Thursday = “” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Sunny14 |
| Test Purpose | To verify that the user can properly register changes to their classes to the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select: 4555  Start Time Tuesday =”1500”  End Time Tuesday = “1700”  Start Time Thursday =”1500”  End Time Thursday = “1700” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Sunny15 |
| Test Purpose | To verify that the user can properly register changes to their classes to the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select 4555  Semester: “Spring”  Start Date = “01/15/19”  End Date = “05/10/19” |
| Expected output | The app closes “Schedule Setup” |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Rainy16 |
| Test Purpose | To verify that the user can’t change certain attributes when updating the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select 4555  Course Name = “Prin. Software Testing” |
| Expected output | The app closes “Schedule Setup” and no changes are made to course\_name of the selected class. |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Rainy17 |
| Test Purpose | To verify that the user can’t change certain attributes when updating the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select: 4555  Semester: “Summer A”  Start Date = “05/20/19”  End Date = “07/28/19” |
| Expected output | The app closes “Schedule Setup” and no changes are to Semester for the selected class |

|  |  |
| --- | --- |
| Test ID | System\_EditClass\_Rainy18 |
| Test Purpose | To verify that the user can’t change certain attributes when updating the database |
| Test Setup | Reinitialize the database to its initial state  Login as:   * user: “Pclarke” * Password: “1234” |
| Test Input | Click “Edit Schedule”  Select: 4555  Course Number = “4073” |
| Expected output | The app closes “Schedule Setup” and no changes are made to the course\_id of the selected class. |

**5.2 Actual Test Results**

|  |  |
| --- | --- |
| Case ID | Result |
| System\_Login\_Sunny01 | Pass |
| System\_Login\_Sunny02 | Pass |
| System\_Login\_Sunny03 | Pass |
| System\_Login\_Rainy04 | Pass |
| System\_Login\_Rainy05 | Pass |
| System\_Login\_Rainy06 | Pass |
| System\_AddClass\_Sunny07 | Pass |
| System\_AddClass\_Sunny08 | Pass |
| System\_AddClass\_Sunny09 | Pass |
| System\_AddClass\_Rainy10 | Pass |
| System\_AddClass\_Rainy11 | Fail |
| System\_AddClass\_Rainy12 | Fail |
| System\_EditSchedule\_Sunny13 | Pass |
| System\_EditSchedule\_Sunny14 | Pass |
| System\_EditSchedule\_Sunny15 | Pass |
| System\_EditSchedule\_Rainy16 | Pass |
| System\_EditSchedule\_Rainy17 | Pass |
| System\_EditSchedule\_Rainy18 | Pass |

**6. Test Summary Report**

This sections details documentation of test cases that did not produce expected output and failed to meet test criteria. We give observations and propose solutions for them.

**Unit Testing**

|  |  |
| --- | --- |
| Test Case ID: | Unit\_test\_loggedin\_Rainy\_030 |
| Test Purpose | Test if the loggedin method, containing the login logic for main, correctly performs login operations |
| Observations: | Loggedin variable is set by the mocked method, but only after manually setting it in the method, otherwise test fails |
| Proposed Solutions | Make changes to the code to so that it can be dynamically tested with mocks |

|  |  |
| --- | --- |
| Test Case ID: | Unit\_Test\_connect\_Rainy\_032 |
| Test Purpose | Test if the connect method correctly stablishes a connection with the database |
| Observations: | Even though we’re passing a null connection, the method does not fail to connect |
| Proposed Solutions | We suspect the connection is being set before hand by some other test. Isolation of this test could be a solution |

|  |  |
| --- | --- |
| Test Case ID: | Unit\_Test\_disconnect\_Rainy\_034 |
| Test Purpose | Test if the disconnect method correctly disconnects from the database |
| Observations: | We did not find a way to accurately mock a disconnection from the database |
| Proposed Solutions | We could try changing the code to mimic a disconnection |

|  |  |
| --- | --- |
| Test Case ID: | Unit\_Test\_storeClassInfo\_Rainy\_076 |
| Test Purpose | Test if the storeClassInfo method correctly stores the class info into database. |
| Observations: | Method stores the wrong class info |
| Proposed Solutions | We suspect the info is being set by another mocked method beforehand. Isolation of this test could be a solution. |

**Subsystem Testing**

|  |  |
| --- | --- |
| Test Case ID: | LogicFacadeTest\_Subsys\_Sunny003 |
| Purpose: | This test was made to see if the appController method is able to return an array of strings from all the method getData called in LogicFacade. |
| Observations: | The methods called did return the correct string that was expected but failed the Junit test, with an error saying the input given did not match the value received. |
| Proposed Solution: | Use Mockito to make sure the outputs match to pass the Junit test case. |

|  |  |
| --- | --- |
| Test Case ID: | courseSelectTest\_Subsys\_Rainy001 |
| Purpose: | Verify that getData is able to return a null when there is no course info available. |
| Observations: | The methods called did return a null but the expected did not match the given, so the Junit test failed. |
| Proposed Solution: | Use Mockito to make sure the outputs match to pass the Junit test case. |

**System Testing**

|  |  |
| --- | --- |
| Test Case ID(s): | System\_AddClass\_Rainy11 |
| Purpose: | To verify that the user can’t register a class with an incorrect value to the database. |
| Observations: | The app closes the schedule setup window, despite an invalid input is given to the program, and it does not add the class to the database. |
| Proposed Solution: | Have the schedule setup form remain open when the save button is pressed so that the user knows that he entered an invalid value. |

|  |  |
| --- | --- |
| Test Case ID(s): | System\_AddClass\_Rainy12 |
| Purpose: | To verify that the user can’t register a class with an incorrect value to the database. |
| Observations: | The app closes the schedule setup window, despite an invalid input is given to the program, and it saves the class information with an invalid input to the database. |
| Proposed Solution: | The system should check if all dates or times entered are valid when the user hits the save button and prevent the schedule setup form from closing­­­­­­. |

**7. Risks and Contingencies**

There are number of potential risks that could arise during testing the PSM software. These are as follows:

1. Scheduling conflicts between team members due to other class projects, work responsibilities, etc. could result in the team having difficulties in setting up meetings.
2. Failure to properly manage what time members did have available, which could result in failure to complete the project and/or poor overall project quality.
3. Poor quality of PSM project, specifically poor readability (lack of comments) and high redundancy (large amount of redundant and unused code), could be an impediment to the team’s comprehension of how the software is supposed to work, which could lead to difficulties in testing.
4. Lack of knowledge/understanding of the tools intended to be used for testing could result in failure to complete the project on time.

Contingency plans for the aforementioned risks are:

1. Establish weekly meeting times that are acceptable for all team members early in the process.
2. Team should strive to start work on the project early and establish deadlines for the completion of specific testing “milestones”. Scrum-style bi-weekly standups could be used to assess each member’s progress, identify and resolve impediments, and ultimately ensure that deadlines are met.
3. A portion of meeting time shall be set aside for the entire group to analyze the PSM software code so that an understanding of the code’s functionality can be gained. Special temporary drivers could be written to serve as aides in understanding.
4. Members should be queried as to which, if any, of the intended testing tools they know how to use. Those team members who do not know any of the tools should be assigned one tool to learn and be given a reasonable amount of time to learn how to use it. If any member knows how to use any of the tools, they should help teach the rest of the team.

**8. Approvals**

|  |  |  |
| --- | --- | --- |
| **Approved By** | **Signature** | **On (Date)** |
| **Raul Espinosa** | *Raul Espinosa* | 2/21/19 |
| **Alejandro Perez** | *Alejandro Perez* | 2/21/19 |
| **David Rocha** | *David Rocha* | 2/21/19 |
| **Samuel Yorizzo** | *Samuel Yorizzo* | 2/21/19 |

**9. Glossary**

JUnit: Framework used for unit and subsystem testing in the Java Programming Language. Can be used in tandem with the Eclipse IDE.

IDE: Integrated Development Environment. A software environment that provides a series of facilities and tools to computer programmers for the purpose of software development.

RFT: Rational Functional Tester. A software tool developed by IBM that is used for automated system testing. It mimics the actions and assessments of a human tester.

Mockito: A framework used for mocking in Java unit testing. Can be used in conjunction with JUnit.

FIU: Florida International University.

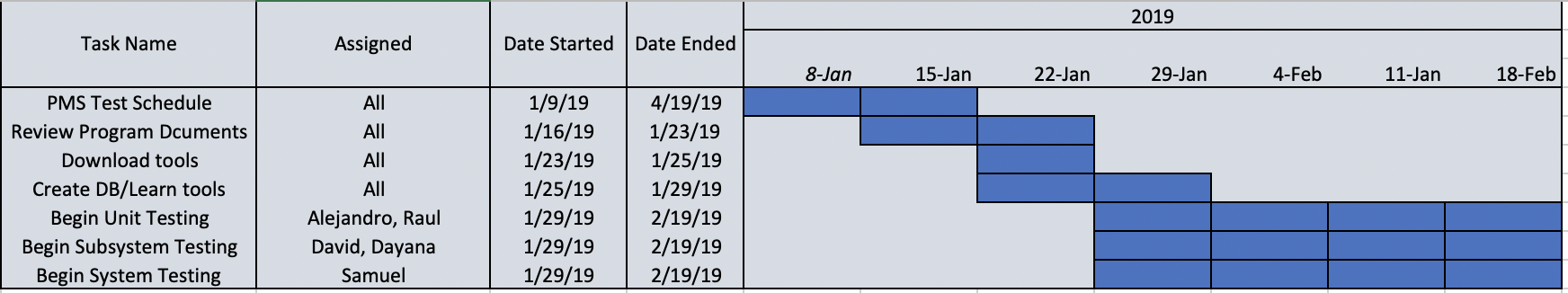
JRE: Java Runtime Environment.

MySQL: A relational database and database management system.

Mocking: The act of creating an instance of an object with discrete values using Mockito.

**10. Appendix**

**10.1 Appendix A – Test schedule**

****

*Figure 10.1.1: Gantt Chart illustrating test schedule*

**10.2 Appendix B – Use Cases (implemented only)**

****

*Figure 10.2.1: Use cases implemented by the PSM system*

**10.3 Appendix C – Example of well documented test drivers, stubs used during unit and subsystem testing**

**Unit Testing**

**Driver**

public class testAppController {

@Mock

Connection;

@Mock

Statement statement;

@Mock

DBConnection dbCon;

@Mock

InterfaceController icMock;

@Mock

Authenticate authMock;

@Mock

LoginForm logMock;

@Mock

Messages msgMock;

appController controller;

@Before

public void setUp() throws Exception {

controller = new appController();

dbCon = mock(DBConnection.class);

statement = mock(Statement.class);

connection = mock(Connection.class);

authMock = mock(Authenticate.class);

icMock = mock(InterfaceController.class);

logMock = mock(LoginForm.class);

msgMock = mock(Messages.class);

controller.setDB(dbCon);

controller.setAuthenticate(authMock);

controller.setInterfaceController(icMock);

}

@After

public void tearDown() throws Exception {

controller = null;

}

}

**Test Case**

@Test

public void testCheckTimesSunny() {

ArrayList<Integer> date = new ArrayList<Integer>();

date.add(1101);

String defSub = "ENC";

        String defSemester = "Spring";

        String defCourseName = "English";

        String defCourseStart = "01/08/19";

        String defCourseEnd = "05/01/19";

        String defMonStart = "15:00";

        String defMonEnd = "16:00";

        String defTueStart = "15:00";

        String defTueEnd = "16:00";

        String defWedStart = "15:00";

        String defWedEnd = "16:00";

        String defThuStart = "15:00";

        String defThuEnd = "16:00";

        String defFriStart = "15:00";

        String defFriEnd = "16:00";

        String defSatStart = "15:00";

        String defSatEnd = "16:00";

when(dbCon.getCourses()).thenReturn(date);

when(dbCon.fetchCourseSubj(1101)).thenReturn(defSub);

when(dbCon.fetchCourseSemester(1101)).thenReturn(defSemester);

when(dbCon.fetchCourseName(1101)).thenReturn(defCourseName);

when(dbCon.fetchCourseStart(1101)).thenReturn(defCourseStart);

when(dbCon.fetchCourseEnd(1101)).thenReturn(defCourseEnd);

when(dbCon.fetchStartMon(1101)).thenReturn(defMonStart);

when(dbCon.fetchEndMon(1101)).thenReturn(defMonEnd);

when(dbCon.fetchStartTue(1101)).thenReturn(defTueStart);

when(dbCon.fetchEndTue(1101)).thenReturn(defTueEnd);

when(dbCon.fetchStartWed(1101)).thenReturn(defWedStart);

when(dbCon.fetchEndWed(1101)).thenReturn(defWedEnd);

when(dbCon.fetchStartThu(1101)).thenReturn(defThuStart);

when(dbCon.fetchEndThu(1101)).thenReturn(defThuEnd);

when(dbCon.fetchStartFri(1101)).thenReturn(defFriStart);

when(dbCon.fetchEndFri(1101)).thenReturn(defFriEnd);

when(dbCon.fetchStartSat(1101)).thenReturn(defSatStart);

when(dbCon.fetchEndSat(1101)).thenReturn(defSatEnd);

assertFalse(controller.checkTimes());

}

**Subsystem Testing**

Test Driver

package my.PSM.PSM\_Logic;

import java.util.\*;

import java.util.Date;

import java.sql.\*;

import my.PSM.PSM\_Storage.DBConnection;

public class LogicFacade {

appController myCon = new appController();

public LogicFacade() {

myCon.db = myCon.getDB();

}

//Facade for getData method

public ArrayList<String> getDataFacade(int course) {

ArrayList<String> courseData = new ArrayList<>();

appController.getData(course);

courseData.add(appController.defSub);

courseData.add(appController.defSemester);

courseData.add(appController.defCourseName);

courseData.add(appController.defCourseStart);

courseData.add(appController.defCourseEnd);

courseData.add(appController.defMonStart);

courseData.add(appController.defMonEnd);

courseData.add(appController.defTueStart);

courseData.add(appController.defTueEnd);

courseData.add(appController.defWedStart);

courseData.add(appController.defWedEnd);

courseData.add(appController.defThuStart);

courseData.add(appController.defThuEnd);

courseData.add(appController.defFriStart);

courseData.add(appController.defFriEnd);

courseData.add(appController.defSatStart);

courseData.add(appController.defSatEnd);

return courseData;

}

//Facade for sleep method

public long sleepFacade(int time) {

long start = System.currentTimeMillis();

long end;

appController.sleep(time);

end = System.currentTimeMillis();

return end - start;

}

//Facade for setTime method

public String setTimeFacade(int year, int month, int date, int hours, int mins) {

myCon.setTime(year, month, date, hours, mins);

return myCon.getTime().toString();

}

//Facade for getTime method

public String getTimeFacade() {

return myCon.getTime().toString();

}

//Facade for getTime method

public Long getTimeInMillisFacade() {

return appController.getTimeMillis();

}

//Facade for timerParser

public String timerParser(String timer) {

appController.timerParser(timer);

String time = myCon.returnHr() + ":" + myCon.returnMin();

return time;

}

//Facade return Hr/Min

public int returnHrFacade() {

return myCon.returnHr();

}

public int returnMinFacade() {

return myCon.returnMin();

}

//Facade for getEndTime

public String getEndTimeFacade(int hrs, int mins) {

String time = appController.getEndTime(hrs, mins).toString();

return time;

}

//Facade for setSemesterClear

public String setSemesterClearFacade(int year, int month, int date, int hrs, int mins) {

appController.setSemesterClear(year, month, date, hrs, mins);

return myCon.getSemesterClear().toString();

}

//Facade for getSemesterClear

public String getSemesterClearFacade() {

return myCon.getSemesterClear().toString();

}

//Facade for get15beforeEnd

public String get15BeforeEndFacade(int hrs, int mins) {

return appController.get15BeforeEnd(hrs, mins).toString();

}

//Facade for get5beforeEnd

public String get5BeforeEndFacade(int hrs, int mins) {

return appController.get5BeforeEnd(hrs, mins).toString();

}

/\*

public void start() {

if(!myCon.getLoginStatus()) {

System.out.println("not logged in");

}

myCon.db = myCon.getDB();//new DBConnection();

//appController.LogIn();

if(myCon.getLoginStatus()) {

System.out.println("logged in");

}

// Test the getData() method using specific course ID

appController.getData(4555);

//Test the sleep() method using 10 secs

//System.out.println("Start sleep");

//appController.sleep(10000);

//System.out.println("End sleep");

//Testing setTime

myCon.setTime(2019,1,14,10,30);

//Testing getTime

System.out.println(myCon.getTime());

//gettimemillis

System.out.println(appController.getTimeMillis());

//timerparser test

appController.timerParser(myCon.defMonEnd);

//return hour/min tests

System.out.println(myCon.returnHr());

System.out.println(myCon.returnMin());

//getEndTime test

System.out.println(appController.getEndTime(10, 30));

//setSemesterClear

appController.setSemesterClear(2019, 1, 14, 12, 30);

//getSemesterClear

System.out.println(myCon.getSemesterClear());

//get15MinsBeforeEnd

System.out.println(appController.get15BeforeEnd(10, 30));

//get5MinsBeforeend

System.out.println(appController.get5BeforeEnd(10, 30));

}

public static void main(String[] args) {

LogicFacade test = new LogicFacade();

}

}

Test Case

package my.PSM.PSM\_Logic;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import my.PSM.PSM\_Storage.DBConnection;

class LogicFacadeTest\_Subsys\_Sunny001 {

private LogicFacade test;

@BeforeEach

void setUp() throws Exception {

test = new LogicFacade ();

}

@AfterEach

void tearDown() throws Exception {

test = null;

}

@Test

void testGetDataFacade() {

// Test String to compare

String array [] = {"ENC", "Spring", "English", "01/08/19", "05/01/19", "15:00", "16:00",

"15:00", "16:00", "15:00", "16:00", "15:00", "16:00", "15:00", "16:00", "15:00", "16:00"};

// Test

assertEquals(array,test.getDataFacade(1101));

}

}

**10.4 Appendix D – Screen shots of the GUI testing tools used for one system test case.**

**A screenshot of a computer screen

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**10.5 Appendix E – Diary of meeting and tasks**

|  |  |
| --- | --- |
| Meeting Date: | 01/17/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 9:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Assign roles.  Pick meeting time.  Assign preliminary tasks. |
| Summary of Discussion: | Team discussed roles appropriate for each member. Roles assigned.  Team discussed meeting date/time all team members are comfortable with. Meeting time set for Thursdays from 7:50 to 9:00pm.  Preliminary tasks were discussed for this week and for the following week. Some members had already done some of the discussed tasks.  Tasks were design so that the entire team was caught up and on the same page. |
| Assigned Tasks: | Preliminary tasks assigned for all members:  Download artifact and go over documents.  Get program running on eclipse environment.  Download RFT and JUnit. |

|  |  |
| --- | --- |
| Meeting Date: | 01/24/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 9:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Review progress.  Review problems encountered in completing tasks assigned.  Assign further tasks depending on progress. |
| Summary of Discussion: | Team discussed progress on previously assigned tasks.  All members were able to get the program running on Eclipse.  Issues with the database were discussed and task was assigned to get the database running.  Refactoring and facades were discussed.  Issues were discussed about installing RFT. Members discussed the issues and progress was made.  Further tasks were discussed for this week and potential task for the following week. |
| Assigned Tasks: | All members were assigned task:  Get the database running and be able to access all parts of project.  Start watching RFT and JUnit tutorials to start getting familiarized with the tools.  Start working on refactoring and facades. |

|  |  |
| --- | --- |
| Meeting Date: | 01/31/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 9:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Review progress.  Get Database running  Refactoring |
| Summary of Discussion: | Database was discussed  Refactoring and facades were discussed.  Work was started on Database and Facades. |
| Assigned Tasks: | Begin testing  Continue familiarizing with the tools. |

|  |  |
| --- | --- |
| Meeting Date: | 02/5/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 9:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on tests  Set testing tools  Set JUnit and Mockito |
| Summary of Discussion: | Testing tools were discussed |
| Assigned Tasks: | Continue testing  Continue familiarizing with the tools. |

|  |  |
| --- | --- |
| Meeting Date: | 02/12/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 9:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on tests |
| Summary of Discussion: | Testing was discussed |
| Assigned Tasks: | Continue testing  Continue familiarizing with the tools. |

|  |  |
| --- | --- |
| Meeting Date: | 02/14/19 |
| Location: | ECS 241 |
| Start time: | 7:50pm |
| End time | 10:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on tests |
| Summary of Discussion: | Testing was discussed |
| Assigned Tasks: | Continue testing  Continue familiarizing with the tools. |

|  |  |
| --- | --- |
| Meeting Date: | 02/16/19 |
| Location: | ECS 241 |
| Start time: | 1:50pm |
| End time | 8:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on tests  Work on documentation  Work on presentation |
| Summary of Discussion: | Testing was discussed  Documentation was discussed  Presentation was discussed |
| Assigned Tasks: | Continue testing  Start Documentation  Start Presentation |

|  |  |
| --- | --- |
| Meeting Date: | 02/18/19 |
| Location: | ECS 241 |
| Start time: | 1:50pm |
| End time | 11:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on tests |
| Summary of Discussion: | Testing was discussed  Documentation was discussed  Presentation was discussed |
| Assigned Tasks: | Continue testing  Continue Documentation  End Presentation |

|  |  |
| --- | --- |
| Meeting Date: | 02/21/19 |
| Location: | ECS 241 |
| Start time: | 1:50pm |
| End time | 11:00pm |
| Attended: | David, Raul, Samuel, Alejandro |
| Late: | none |
| Agenda: | Work on documentation |
| Summary of Discussion: | Documentation was discussed |
| Assigned Tasks: | Finish Documentation |