

**NANYANG**  
**TECHNOLOGICAL**  
**UNIVERSITY**

## **Automatic Assessment System for Java Programs**

### **Final Report**

**Submitted By: Huang Jian Wei**

**Matriculation No: U1521567A**

**Supervisor: Mr. Tan Kheng Leong**

**Examiner: Assoc Prof Deepu Rajan**

A final year project report presented to the Nanyang Technological University  
in partial fulfillment of the requirements of the degree of  
Bachelor of Computer Science and Engineering

**AY 2017/2018**

**SCHOOL OF COMPUTER SCIENCE AND  
ENGINEERING NANYANG TECHNOLOGICAL  
UNIVERSITY**

## Abstract

Automatic evaluation and assessment systems have been a growing topic of interest. Studies and research have been done to evaluate the limitations of traditional assessment methods and how the use of technology can overcome these barriers to simplify the learning process of students in universities.

In the field of computing, students' knowledge and programming techniques are typically assessed through their lab sessions and quizzes, as well as assignments. With ever-growing students' intake every year, it can be challenging and time-consuming for academic staff to access each student's program individually, leading to excessive workload.

This project proposes a system where Java programs can be assessed automatically and accurately, hence significantly reducing the time needed for academic staffs to spend on manually marking such programs. Additionally, the system can also provide immediate feedback to the student in the form of a summarized report, with scores based on how the student fared and a summary of errors, if any.

The deployment of this system would be a web-based application with a backend database and server support, as most of the heavy processing should be offloaded to the server.

This paper will focus on the implementation and limitation of the current system developed, its functionality and the future improvements that can be done.

## Acknowledgments

The author would like to express his utmost gratitude and appreciation to his supervisor, Mr. Tan Kheng Leong, for providing guidance and suggestions throughout the Final Year Project period. Mr. Tan Kheng Leong's willingness in allowing the author to try out and experiment different ideas in implementing the project has allowed the author to gain invaluable experience and broadening his horizons. The author would also like to express his appreciation for Mr. Tan Kheng Leong's effort in checking up on his progress every few weeks and providing helpful insights for the project.

The author is also extremely thankful for the support of his family and close friends for their encouragement and support throughout the duration of this project.

## TABLE OF CONTENTS

<b>INTRODUCTION .....</b>	<b>9</b>
<u>1.1 Motivation .....</u>	9
<u>1.2 Review of existing automatic assessment systems.....</u>	10
<u>1.3 Project Objective and scope.....</u>	12
<u>1.3.1 Functional Requirement.....</u>	13
<u>1.3.2 Non-Functional Requirement.....</u>	14
<u>1.4 Use Case.....</u>	16
<u>1.5 Stakeholders.....</u>	21
<u>1.6 Assumptions and Constraints.....</u>	21
<u>1.7 Project Schedule.....</u>	22
<u>1.8 Report Organization.....</u>	23
<b>2 REQUIREMENT ANALYSIS .....</b>	<b>24</b>
<u>2.1 Project Overview.....</u>	24
<u>2.2 Activity Diagram.....</u>	26
<u>2.2.1 Student Activity Diagram.....</u>	26
<u>2.2.2 Admin Activity Diagram.....</u>	27
<u>2.3 Development Tools.....</u>	28
<u>2.4 Development Enviroment.....</u>	28
<u>2.4.1 Hardware specification.....</u>	28
<u>2.4.2 Software specification.....</u>	28
<b>3 DESIGN .....</b>	<b>29</b>
<u>3.1 User Interface Design.....</u>	29
<u>3.2 System Design Principles.....</u>	31
<u>3.3 System Architecture.....</u>	32
<u>3.3.1 RESTful Client-Server Architecture.....</u>	33
<u>3.3.2 Presentation Layer .....</u>	33
<u>3.3.3 Application Layer.....</u>	35
<u>3.3.4 Data layer.....</u>	38
<u>3.4 Database Design.....</u>	38
<u>3.4.1 Database Configuration.....</u>	38
<u>3.4.2 Entity Relationship Diagram.....</u>	39
<u>3.3.3 Database Tables.....</u>	39
<u>3.5 Summary of Tools and technologies used.....</u>	46
<b>4 SYSTEM IMPLEMENTATION .....</b>	<b>47</b>
<u>4.1 System Functions.....</u>	48
<u>4.1.1 Login Page.....</u>	51

<u>4.2 Student Functions Implementation.....</u>	<u>51</u>
4.2.1 Student Home Page .....	51
4.2.2 Attempt Assessment Question - Log in Session.....	53
4.2.3 Attempt Assessment Question - Save and compile Question.....	56
4.2.4 Attempt Assessment Question - Save and Submit Solution.....	59
4.2.5 Attempt Assessment Question - Compute Score and Generate Report.....	60
4.2.6 View Labs.....	64
4.2.7 View Past Attempts.....	66
<u>4.3 Admin Functions Implementation.....</u>	<u>67</u>
4.3.1 Create Quiz Session.....	68
4.3.2 Delete Quiz Session.....	68
4.3.3 Modify Quiz Questions.....	69
4.3.4 Create Assessment Questions.....	69
4.3.5 Add Students.....	71
4.3.6 View Student Attempts.....	72
<b>5 LIMITATION AND DIFFICULTY FACED.....</b>	<b>73</b>
<b>6 CONCLUSION AND FUTURE RECOMMENDATIONS.....</b>	<b>74</b>
6.1 Conclusion.....	74
6.2 Future Recommendations.....	75
 <b><u>APPENDIX A – DIRECTORY OF PROJECT FILES .....</u></b>	 <b><u>77</u></b>
<b><u>APPENDIX B – FUNCTIONAL TESTING .....</u></b>	<b><u>77</u></b>
<b><u>APPENDIX C – REFERENCES.....</u></b>	<b><u>77</u></b>

## Table of Figures

Figure 1-1: Overview of the Java Automatic Assessment System(JAAS).....	12
Figure 1-2: Use Case Diagram.....	16
Figure 1-3: Project Schedule.....	22
Figure 2-1: 3-Tier Architecture of JAAS.....	24
Figure 2-2: Activity diagram for Student.....	26
Figure 2-3: Activity diagram for Admin.....	27
Figure 3-1: Overview of JAAS's System Architecture.....	32
Figure 3-2: Use of Ng-Repeat directive from AngularJS.....	35
Figure 3-3: Use of Node-cmd to compile files.....	36
Figure 3-4: Configuration of MySQL in config.js.....	38
Figure 3-5: Entity Relation Diagram of JAAS.....	39
Figure 3-6: User Table.....	40
Figure 3-7: Labs Table.....	41
Figure 3-8: Announcement Table.....	41
Figure 3-9: LabSubmissionRecords Table.....	42
Figure 3-10: QuizSubmissionRecords Table.....	43
Figure 3-11: Session Table.....	44
Figure 3-12: User_Session Table.....	44
Figure 3-13: QuizQuestion Table.....	45
Figure 3-14: LabQuestion Table.....	46
Figure 3-15: Summary of Tools and Technologies used.....	46
Figure 4-1: Conceptual Class Diagram.....	47
Figure 4-2: Sequence Diagram – User Login.....	48
Figure 4-3: Sequence Diagram – Lab Submission.....	48
Figure 4-4: Sequence Diagram - Quiz Submission.....	49
Figure 4-5: Sequence Diagram – Create Assessment Question.....	49
Figure 4-6: Sequence Diagram – Delete Student.....	50
Figure 4-7: Sequence Diagram – View Student Attempts.....	50
Figure 4-8: Successful Login.....	51
Figure 4-9: Unsuccessfully Login.....	51
Figure 4-10: Student Home Page displaying announcements .....	51

SCE17-0389  
Java Program Auto Assessment

Figure 4-11: Displaying Announcements using AngularJS binding.....	52
Figure 4-12: POST request retrieving announcements from the database.....	52
Figure 4-13: Server-side query to retrieve announcements.....	52
Figure 4-14: Login to Assessment Session.....	53
Figure 4-15: Assessment Session Page.....	53
Figure 4-16: Function to generate questions for Assessment Session.....	54
Figure 4-17: Example of User_session record.....	55
Figure 4-18: Client-side request for questions data.....	55
Figure 4-19: Retrieving question title and question description from database.....	55
Figure 4-20: Displaying the Assessment questions page.....	56
Figure 4-21: .java and .class file generated based on student's submitted codes.....	57
Figure 4-22: Codes to write student's code to .java file.....	58
Figure 4-23: Unsuccessful attempt at compiling a file due to syntax errors.....	58
Figure 4-24: Attempting submission of solution.....	59
Figure 4-25: Relationship between a question and a question configuration file.....	60
Figure 4-26: Example: Content of a question configuration file.....	60
Figure 4-27: Process of assessing a student's program.....	61
Figure 4-28: Listening and capture the stdout/output of a student's program.....	61
Figure 4-29: Code to simulate user input.....	62
Figure 4-30: Input and Output of student's program being displayed in console.....	62
Figure 4-31: Code to check for keywords in student's code.....	62
Figure 4-32: Keywords checked vs keywords expected displayed in the terminal.....	62
Figure 4-33: Content of a report generated by the system for the student.....	63
Figure 4-34: View Lab Page.....	64
Figure 4-35: Displaying Lab document to student.....	64
Figure 4-36: UI for submission of lab assignments .....	65
Figure 4-37: User uploading a file for submission.....	65
Figure 4-38: Error message shown in the report.....	66
Figure 4-39: View past attempts of student's lab and quiz submissions.....	66
Figure 4-40: Viewing past report online.....	67
Figure 4-41: User Interface for creating sessions.....	68
Figure 4-42: Code: Writing session details into database.....	68

SCE17-0389  
Java Program Auto Assessment

Figure 4-43: User Interface to delete quiz.....	68
Figure 4-44: Modify Session User Interface.....	69
Figure 4-45: Creating a Quiz assessment question.....	69
Figure 4-46: Creating a Lab assessment question.....	70
Figure 4-47: Error message when testing an assessment question before creation.....	71
Figure 4-48: User Interface for adding students.....	71
Figure 4-49: Excel format to upload student list.....	72
Figure 4-50: Viewing student's past attempts.....	72

## Table of Figures - Appendix

Figure A-1: Files and folder in the root directory.....	77
Figure A-2: Sub Folder in App Folder.....	77
Figure A-3: Views Directory.....	78
Figure A-4: Admin Views.....	78
Figure A-5: Student Views.....	78
Figure A-6: Controller Directory.....	79
Figure A-7: Model Directory.....	79
Figure A-8: Content of package.json, list of packages used.....	80
Figure A-9: Web server configurations.....	81
Figure A-10: POST URL.....	82
Figure A-11: POST Request returning a resource file.....	82
Figure A-12: Starting the Web Application using nodemon in console .....	82



# 1. Introduction

## 1.1 Motivation

The world has seen a rapid technological advancement in the past decade, and the demand for knowledge-based workers in Information Technology (IT) industry is ever growing [1]. This will result in more students will be enrolling in an IT-related course in time to come, and programming is the core of such courses. A typical IT student is assessed based on three main components: lab sessions, assignments, and a final exam which will summarize what the student has learned throughout the course. Lab sessions and assignments allow students to better understand the basic concepts and put into implementation the theories taught. Traditional software assessment methods consist of having academic staff manually reading students' code and running each program individually and testing its output, which has been proven to be very time-consuming [2]. The problem is magnified when the number of students increases, leading to even more excessive workload for the academic staffs.

The use of automatic evaluation and assessment systems have been a growing topic of interest across many research institutions and universities. Research has been done to evaluate the limitations of traditional assessment methods and how the use of technology can overcome these barriers to simplify the learning process of students in educational institutions [3]. The earliest automatic grading system can be dated back to 1960, where such a tool was used for a formal course in programming. [4] Some of the more notable automated assessment systems that have been developed over the years are the TRY system in the late 80's for Unix operating system and Web-CAT (2008), an open source automated grading system that grades the student's code and

assignments by writing their own test cases [4]. These assessment systems can provide almost immediate feedback to students, which is invaluable especially when a program grows in complexity. However, these automated assessment systems always have drawbacks, such as their unawareness towards logical errors caused by the incorrect structuring of codes. At the end of the day, lecturers and professors are still required to examine the student's code to provide accurate advice for the structure and implementation of the program [5].

## 1.2 Review of existing automatic assessment systems

Due to the complexity of programming, there is no single correct solution to assessing programming assignments. Many automatic assessment systems have adopted different strategies to suit the needs of their respective organization and institution [6].

The metrics that were used by these automatic assessment systems to grade these programs include [7]:

- Correctness of code, by comparing the output of student's program with a model output. This is also known as black-box testing where the internal structure of the code does not matter
- Style of code, if the code has been well-commented and indented properly
- Structure of code, by having well-defined modules and procedures (e.g. use of JUnit, white-box testing)
- Efficiency of code, restriction on time complexity (e.g.  $O(n)$  versus  $O(n^2)$ )

As these metrics are not mutually exclusive, it is possible to develop a system that can evaluate all these metrics.

### **Ceilidh-CourseMaster (Computer Environment for Interactive Learning in Diverse Habitats)**

Ceilidh is one of the successful automatic assessment systems that has been developed in the University of Nottingham and installed in various sites across America, Australia, Belgium, China, India, Malaysia, New Zealand, Portugal, Russia and Spain in the early 90s [8]. The main principle of Ceilidh was to use string matching to compare a student's program output against a model output [9]. It was further developed into Ceilidh-CourseMaster in the late 90s. Till date, many of the automatic assessment systems were modeled after Ceilidh due to its proven tracks.

### **Scheme-Robo**

Scheme Robo is an on-line automatic assessment system developed by Helsinki University of Technology in Finland to assess programming exercise written in the functional programming language Scheme [9]. Due to the nature of Scheme, comparison of output against a benchmarked output is not suitable as most exercise requires the student to write single scheme procedure rather than complete programs [9]. Instead, structural analysis is used to check if the student uses the correct internal definitions. This method is similar to JUnit testing.

### **BOSS Online Submission and Assessment System**

BOSS is a tool for assessment of programming assignments which uses two approaches to assess the quality of a student's program. In addition to testing the program for its output, BOSS uses JUnit (for Java programs) to test for the equality of two methods given a specific input [6].

## 1.3 Project Objective and Scope

The aim of this Final Year Project is to develop a fully Java Automated Assessment System(JAAS) that determines the code quality of the codes and programs submitted by students from Nanyang Technological University(NTU) enrolled in the course CZ2006, Object Oriented Design and Programming.

The Java Automated Assessment System provides a web application interface which allows student to upload their codes to the server. Upon successful upload, the server will attempt to compile the student code and spawn an instance of the program if compilation is successful. The server then run test cases against the instance of the program and score the program by verifying its output.

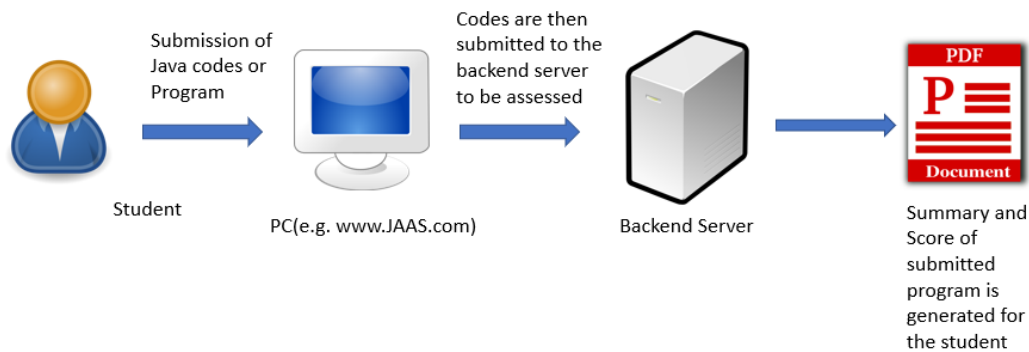


Figure 1-1. Overview of the Java Auto Assessment System(JAAS)

During the study of these existing automatic assessment systems, it is observed that the methodologies used by the developers to assess these students' programs vary based on the objective of the course.

As CZ2006 is an introductory course on Java and Object-oriented programming, it is important to grade students based on the right metrics. JUnit was initially considered, but later rejected as

students are expected to code complete, working applications instead of specific modules which return values. Thus, the correctness of code was chosen to use as the metric for evaluation. (e.g. comparing output against an expected output). In addition, the system will also grade the code based on specific keywords found, such as “extends”, “super” etc. during their exercise on inheritance.

The Java Auto Assessment System must be able to provide the following functional and non-functional requirements:

### 1.3.1 Functional Requirements

- **User Login:** There will be two roles implemented in the system; the student and the admin role. Users must be able to log into the system at any point in time with their given account. An error message should be prompt if login fails.

#### **Student**

- **View Announcements:** Student must be able to view the announcements on the homepage.
- **View and Submit Lab Solutions:** Student must be able to view existing lab.pdf documents and attempt their submission of their solution in the “Labs” tab.
- **Attempt Assessment Questions:** Student must be able to log in an Assessment Session and attempts the questions generated for the assessment question. A report that details the student’s score and description of the error, if any, must be generated.
- **View Past Attempts:** Student must be able to view past attempts of Labs and Assessment Questions. The student must be able to see the date and time of submission and a summarized report which contains the score of the attempt.

## **Admin**

- **Manage Assessment Question:** Admin must be able to add new assessment questions to the system. Admin must also be able to delete or modify existing questions in the system.
- **Manage Assessment Session:** Admin must be able to create, delete or modify an assessment session.
- **View students' Past Attempts:** Admin must be able to see the labs and assessment question attempts of every student in this page. Information regarding the time of submission, author and score must be displayed.
- **Manage Scoring Scheme:** Admin must be able to edit the weightage of scoring scheme. The scoring weightage consists of compilation, syntactical checks, and output correctness.
- **Manage Student Records:** Admin must be able to add, delete or modify students' records in the system. The page must also be able to show a list of all student users in the system
- **Manage Lab:** Admin must be able to add, delete or modify lab documents. These lab documents will be able to be viewed as a pdf file.

### **1.3.2 Non-Functional Requirements**

The Java Auto Assessment System must possess the core system qualities as followed:

#### **Availability**

The system must always be available with no downtime except for the duration of maintenance, as there will be multiple users accessing the system at different point in time. In addition, all information held by the database must be available to be modified at any point in time.

## **Usability**

The design of the user interface must be simple and intuitive. The user interface tries to follow Schneiderman's "Eight Golden Rules of Interface Design" to provide a smooth user experience.

## **Reliability**

All errors and exception cases that will cause the system to crash must be handled. All data and information to be displayed must be accurate and up to date.

## **Security**

As information in the database is highly sensitive, the system must be protected from attacks such as SQL injection(SQLi), Directory Traversal and unrestricted URL access. To prevent and deter such attacks, prepared statements was used to ensure that SQL statements are not modified. Sessions are also used to ensure proper rights and permission control.

## **Performance**

The system must be able to handle multiple user requests at any given time without any loss of performance (e.g. computational time). All request must be responded within a reasonable period of less than one second.

## **Maintainability**

To ensure that future developers can perform maintainability functions and add new features to the system without much difficulty, all codes will be commented and well-documented.

## 1.4 Use case

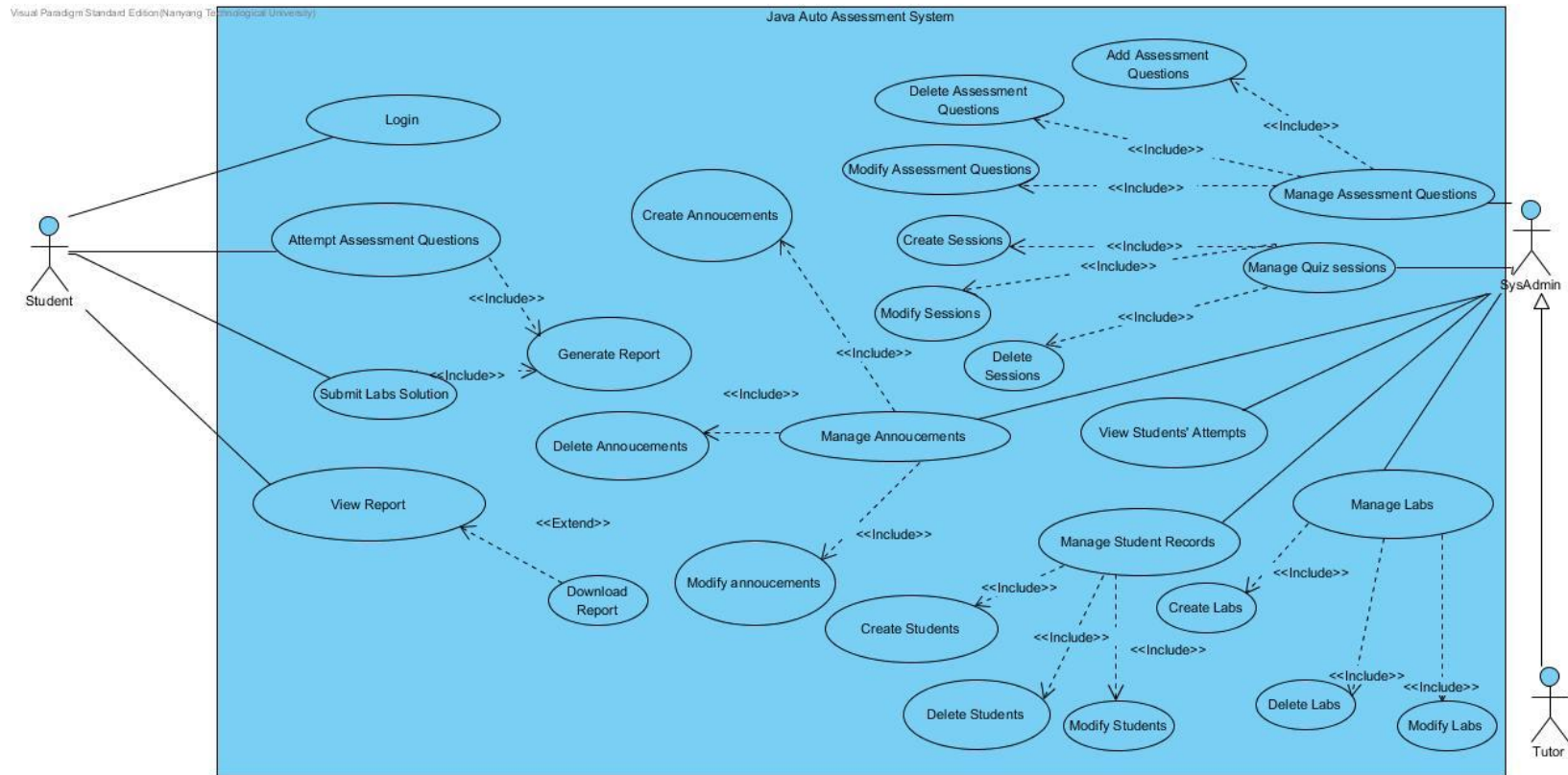


Figure 1-2. Use Case Diagram



The following use case scenarios are of higher complexity and will be described with a use case description to provide a better envision of the system.

Use Case ID:	001		
Use Case Name:	Attempt Assessment Question		
Created By:	Huang Jian Wei	Last Updated By:	Huang Jian Wei
Date Created:	29 May 2017	Date Last Updated:	22 March 2018

Actor:	Student
Description:	Student provides answers to the assessment questions
Preconditions:	Successfully logged in to the system
Postconditions:	Report is generated
Priority:	High
Frequency of Use:	High
Flow of Events:	<p>1.) System randomly generates a set of assessment questions from a collection</p> <p>2.) Student are required to solve the questions using Object Oriented Concepts in Java Programming Language</p> <p>3.) Student enters solution in the text area provided for each question</p> <p>3.1) Student can click Compile button and test solution for result(Success/errors)</p> <p>3.2) Student confirm and submits solution to be assessed by the System.</p> <p>4.) System compiles student-submitted solution and assess according to the scoring scheme</p> <p>5.) Report is generated</p>
Alternative Flow:	System prompt error message while compilation
Exceptions:	<b>EX.1</b> Server could not process the request
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

SCE17-0389  
Java Program Auto Assessment

Use Case ID:	002		
Use Case Name:	Lab Submission		
Created By:	Huang Jian Wei	Last Updated By:	Huang Jian Wei
Date Created:	29 May 2017	Date Last Updated:	22 March 2018

Actor:	Student
Description:	Student uploads a .zip file or .java file on lab submission page
Preconditions:	Successfully logged in to the system
Postconditions:	Report is generated
Priority:	High
Frequency of Use:	High
Flow of Events:	<p>1.) Student chooses and uploads his lab solution</p> <p>2.) Student clicks Submit button</p> <p>3.) System compiles student-submitted files and assess according to the scoring scheme</p> <p>4.) Report is generated</p>
Alternative Flow:	System prompt error message while compilation
Exceptions:	<b>EX.1</b> Server could not process request
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

SCE17-0389  
Java Program Auto Assessment

Use Case ID:	003		
Use Case Name:	Create Assessment Questions		
Created By:	Huang Jian Wei	Last Updated By:	Huang Jian Wei
Date Created:	29 May 2017	Date Last Updated:	22 March 2018

Actor:	Admin
Description:	Admin
Preconditions:	Successfully logged in to the system
Postconditions:	Assessment Question is created
Priority:	High
Frequency of Use:	High
Flow of Events:	<p>1.) Admin chooses between lab question or quiz question</p> <p>2.) Admin enters question title, question description, and the respective weightage for the question.</p> <p>3.) Admin will then be required to either create or upload a question a configuration file. The question configuration file contains the simulated inputs, expected output, and keywords expected from the program.</p> <p>4.) Admin can upload a sample program to test the configuration file if it is valid</p> <p>5.) Admin clicks on create button and question is created</p>
Alternative Flow:	System prompt error while testing the configuration file
Exceptions:	<b>EX.1</b> Server could not process the request
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

SCE17-0389  
Java Program Auto Assessment

Use Case ID:	004		
Use Case Name:	Create Labs		
Created By:	Huang Jian Wei	Last Updated By:	Huang Jian Wei
Date Created:	29 May 2017	Date Last Updated:	22 March 2018

Actor:	Admin
Description:	Admin uploads a lab document for a lab
Preconditions:	Successfully logged in to the system
Postconditions:	Lab document available for view to all student users
Priority:	High
Frequency of Use:	High
Flow of Events:	1.) Admin clicks on “Create Lab” tab  2.) Admin enters lab No, lab title and chooses a lab document in .pdf format  3.) Admin clicks on create button  4.) Lab is created and reflected in the database
Alternative Flow:	-
Exceptions:	<b>EX.1</b> Server could not process request
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

## 1.5 Stakeholders

The primary users of this system will be the students and academic staffs (Lecturers, Professors, Lab staffs etc.) from the School of Computer Science and Engineering in NTU. The role of the students will be submitting codes and programs of their lab sessions, quizzes, and assignment for the system's evaluation.

The academic staff will be the administrators to maintain the system and update the evaluation criteria and the scoring scheme of the system. Administrators can also add in questions for quizzes as well as labs in the system.

## 1.6 Assumptions and Constraints

The following are the assumptions made by the developer during the project's implementation:

1. Deliberate attacks (e.g. DDoS) are not made to the hosting server while the server is live
2. Hosting server has the sufficient hardware and software to handle the tasks required
3. Users are accessing the system using either Firefox, Google Chrome or Internet Explorer web browser
4. Server is hosted on a machine running on Windows Operating System

## 1.7 Project Schedule

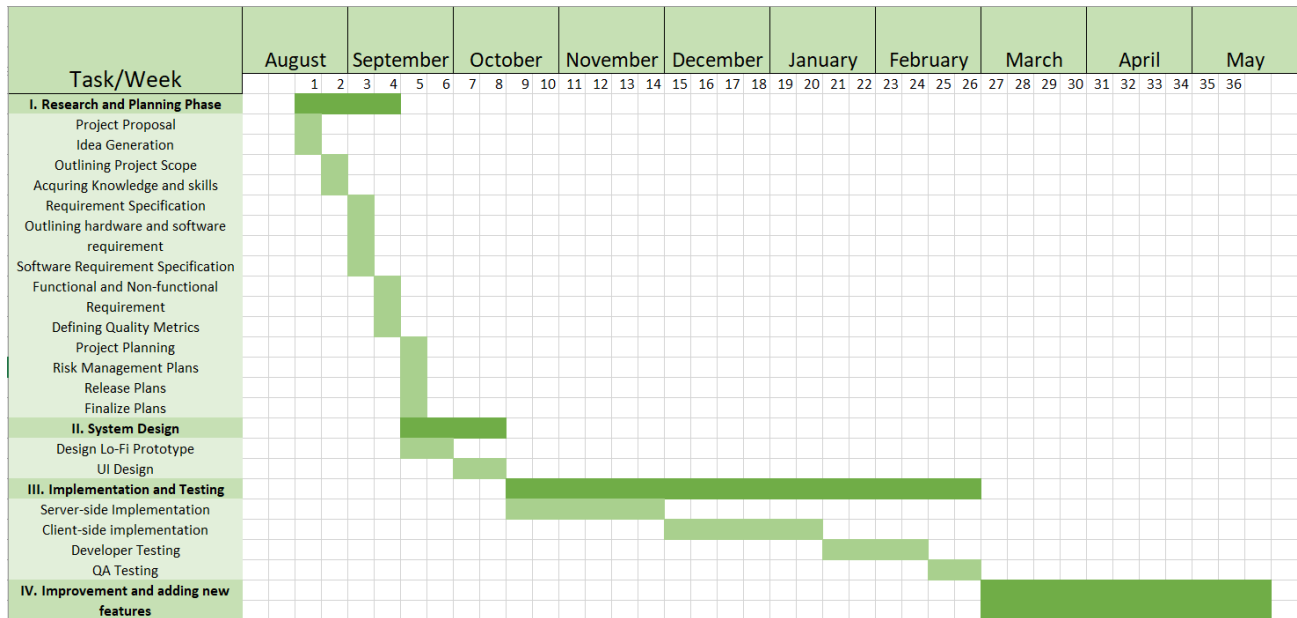


Figure 1-3. Project Schedule of the JAAS

This project will undergo 4 major phases of development. Firstly, sufficient research must be done on existing implementation of such automatic assessment systems. Functional and non-function requirements must be established, and risk management plan must be developed.

Secondly, the system architecture and the user interface must be designed to ensure that the next phase will proceed smoothly.

Thirdly, the project will enter an intensive 20-week implementation stage, where concepts are put into action.

Lastly, heavy emphasis will be placed on debugging and testing the system for live usage.

## 1.8 Report Organization

This report will be organized into 7 sections.

Section 1 provides the background, purpose, scope, and time as well as the timeline for this project.

Section 2 summarizes the development environment, tools, technology used, functional and non-functional requirements.

Section 3 illustrates the system architecture as well as introducing the web stack used. Database design and the entity relationships are also explained in this section.

Section 4 focuses on the implementation of the functions for this project. This chapter also covers the user interface implementation, backend processing, and database implementation.

Section 5 introduces the functional testing applied for the web application

Section 6 concludes the report by summarizing up the major functions, the limitations of the system as well as proposing future enhancement and work to be done.

In addition, there will be many high-level diagrams and segments of codes placed across this report to aid in the understanding of the technical aspect of this project.

## 2. Requirement Analysis

### 2.1 Project Overview

The Java Automated Assessment System(JAAS) is a thin-client web application that consists of three major components; the User Interface(UI) frontend, a backend server which will act as the processing engine and a database which will store persistent data. JAAS also adopts the 3-tier architecture where communication between the UI and the data will go through the controller layer.

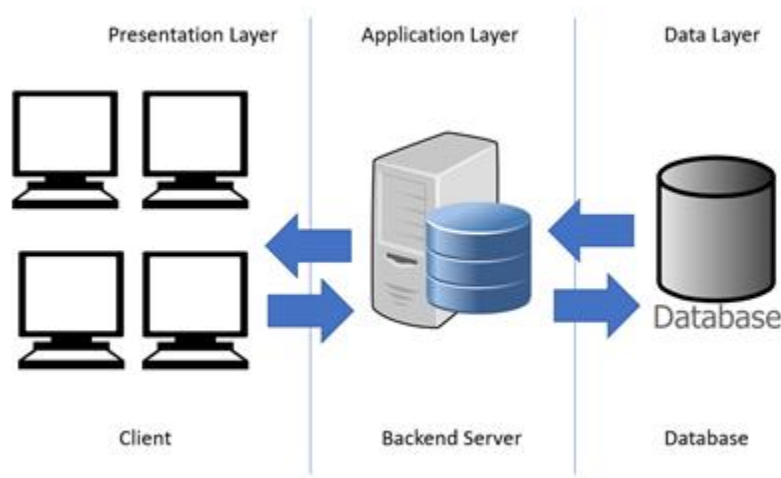


Figure 2-1. 2-tier architecture of JAAS

#### UI Frontend (Presentation Layer)

These will be the webpages that are displaying content to the users in the form of Hyper-Text Transfer Protocol(HTTP), Cascading Stylesheets(CSS) and JavaScript(JS). JAAS leverages on both the AngularJS framework as well as the Bootstrap framework.



### **Server Backend (Application Layer)**

As this web application is thin-client, most of the heavy processing happens at the application server. The application server and business logic utilize NodeJS technology which provides a non-blocking I/O API that optimizes JAAS throughput and scalability.

### **Database (Data Layer)**

The Database Management System(DBMS) used for the stack is MySQL. The database mainly stores persistent data such as user information, assessment questions, and records of student's attempts for their labs, quizzes, and assignments.

## 2.2 Activity Diagram

### 2.2.1 Student

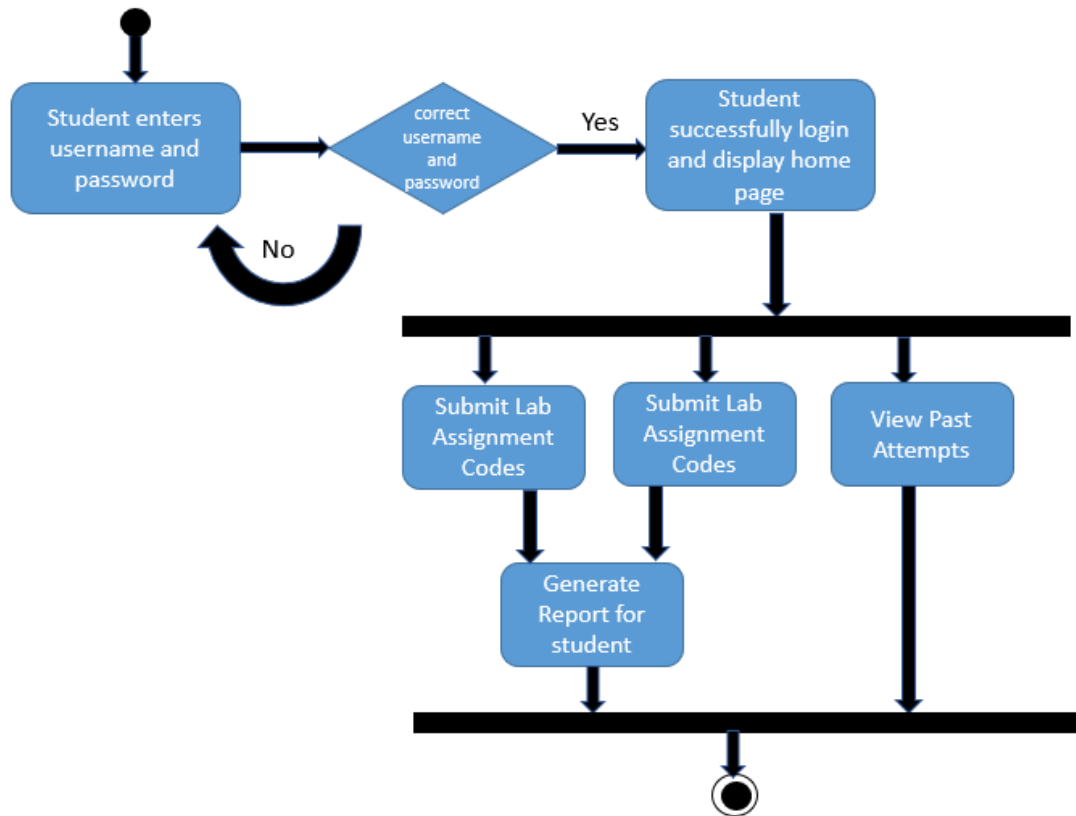


Figure 2-2. Student Activity Diagram

## 2.2.2 Admin

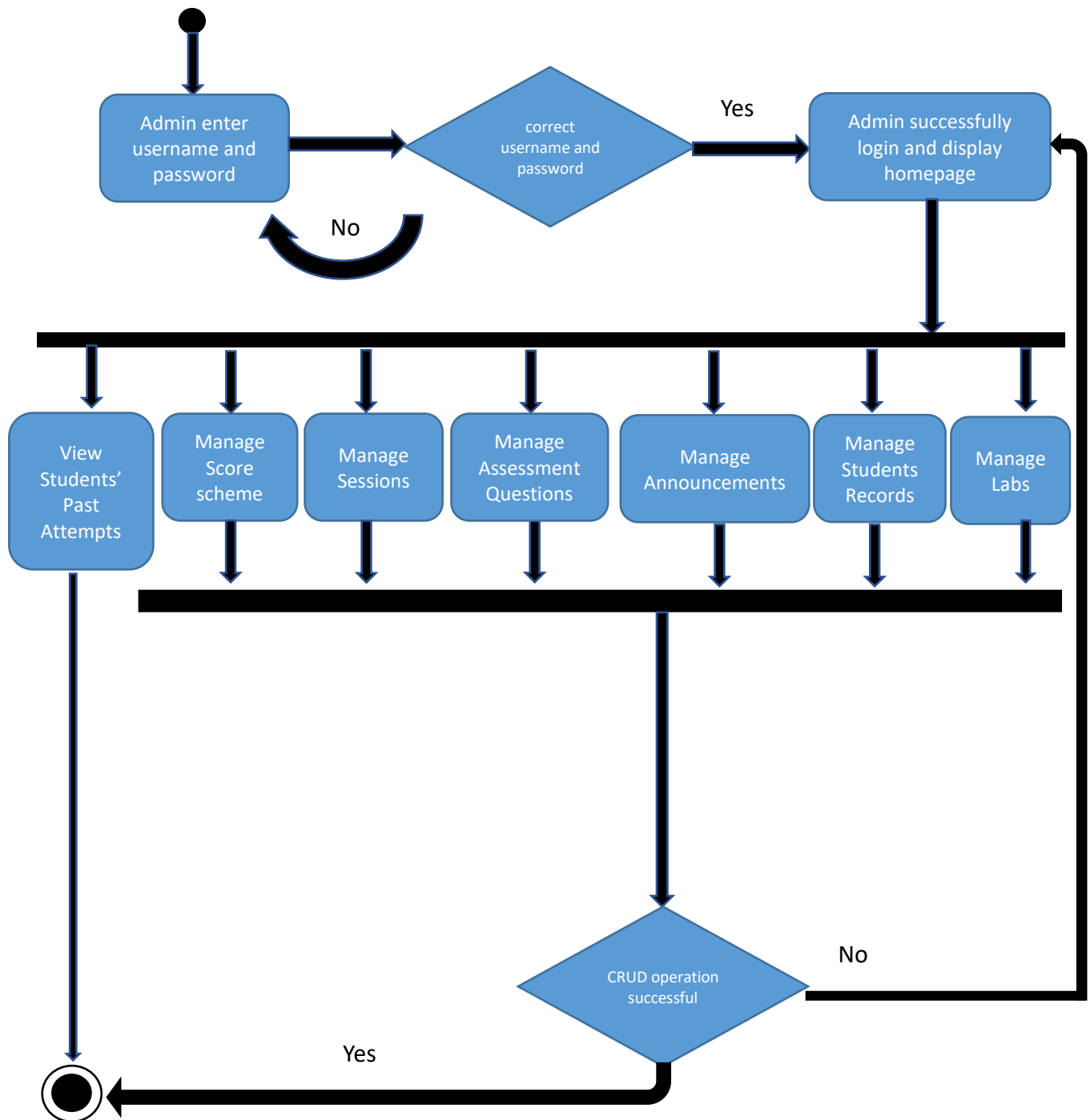


Figure 2-3. Admin Activity Diagram

## 2.3 Development Tools

**Integrated Development Environment(IDE):** Eclipse Oxygen, Sublime Text 3, Notepad++

## 2.4 Development Environment

### 2.4.1 Hardware

**Operating System:** Windows 10 Education

**Processor:** Intel® Core™ i5-6400 CPU @ 2.70GHz 2.70Ghz, 1TB HDD

**Installed memory:** 16GB

### 2.4.2 Software

**Database:** MySQL 5.7

**Front-end:** HTML, Bootstrap, jQuery, JavaScript, AngularJS

**Back-end:** NodeJS/JavaScript

**Java Runtime Environment(JRE):** 1.8.0

**Version Control Software:** Github

**IDE:** Notepad++, Sublime Text 3, Eclipse

**UML Diagram Tool:** Visual Paradigm

**Others:** Dropbox, Google Drive

## 3. Design

This section will focus on the consideration of the system design and architecture.

### 3.1 User Interface Design

The Java Auto Assessment Program(JAAS) follows a simplistic design approach to ensure that every page will not take more than a second to load. In addition to easier navigation, a simplistic design will also mean that maintenance task's difficulty is kept to a minimum. JAAS also follows the guideline and principle of Schneiderman's Eight Golden Rules.

#### **Striving for Consistency**

As students are familiar with NTULearn Blackboard platform, the system is designed in such a way that it has navigation similar to it, allowing students to apply previous navigation knowledge from the Blackboard to the JAAS. An example would be the sequence of action for submission of assignments on the JAAS mirroring the Blackboard.

#### **Use of shortcuts**

Not applicable to the current version of the JAAS. JAAS does not have a task that requires macro or additional keyboard shortcuts.

#### **Offering Informative Feedback**

If a user has performed a task incorrectly, JAAS will prompt a human-readable message to the user to allow the user to understand why his action failed and what steps he should take to achieve his task.

### **Dialogs that yield Closure**

For every action that the user performs, JAAS provides a message to the user so that they will know if their task was successful or unsuccessful. These dialogs will allow users to understand the consequences of their action.

### **Simple Error Handling**

JAAS is designed such that users will not be able to make a serious error in the process of performing their tasks. For minor errors such as incorrect inputs, an error message will be prompt above or below the input field.

### **Permitting Easy Reversal of Action**

In the event that user made a mistake while performing an action, the system must provide ways to reverse these actions. For example, even though an admin has accidentally keyed in a wrong student to be removed, a dialog will be prompted to confirm the action. At this point, if the admin realizes that the input is wrong, he can click Cancel in the dialog to return him to his previous step.

### **Supporting Internal Locus of Control**

To provide users with a sense of control over the system, outcomes, and output of each action must not vary too much from user's expected result. For example, clicking the logout button must immediately log off the user from the system and bring the user back to the login page.

### **Reduce Short-Term Memory Load**

By designing the user interface with a repetitive layout and simplistic navigation buttons, users will not be overloaded with information pertaining to perform a task. By recognizing the way to perform an action rather than recalling from memory, this will provide a sense of satisfaction to the user.

## 3.2 System Design Principle

To design an architecture that minimizes cost and maintenance requirement, as well as promote usability and extensibility [11], JAAS follows a few key principles such as separation of concerns, single responsibility principle, and minimized upfront design. By dividing and separating codes into their respective classes and functions, it will be easier to identify the functions that require changes in the future and limit the area of codes that need to be modified.

### 3.3 System Architecture

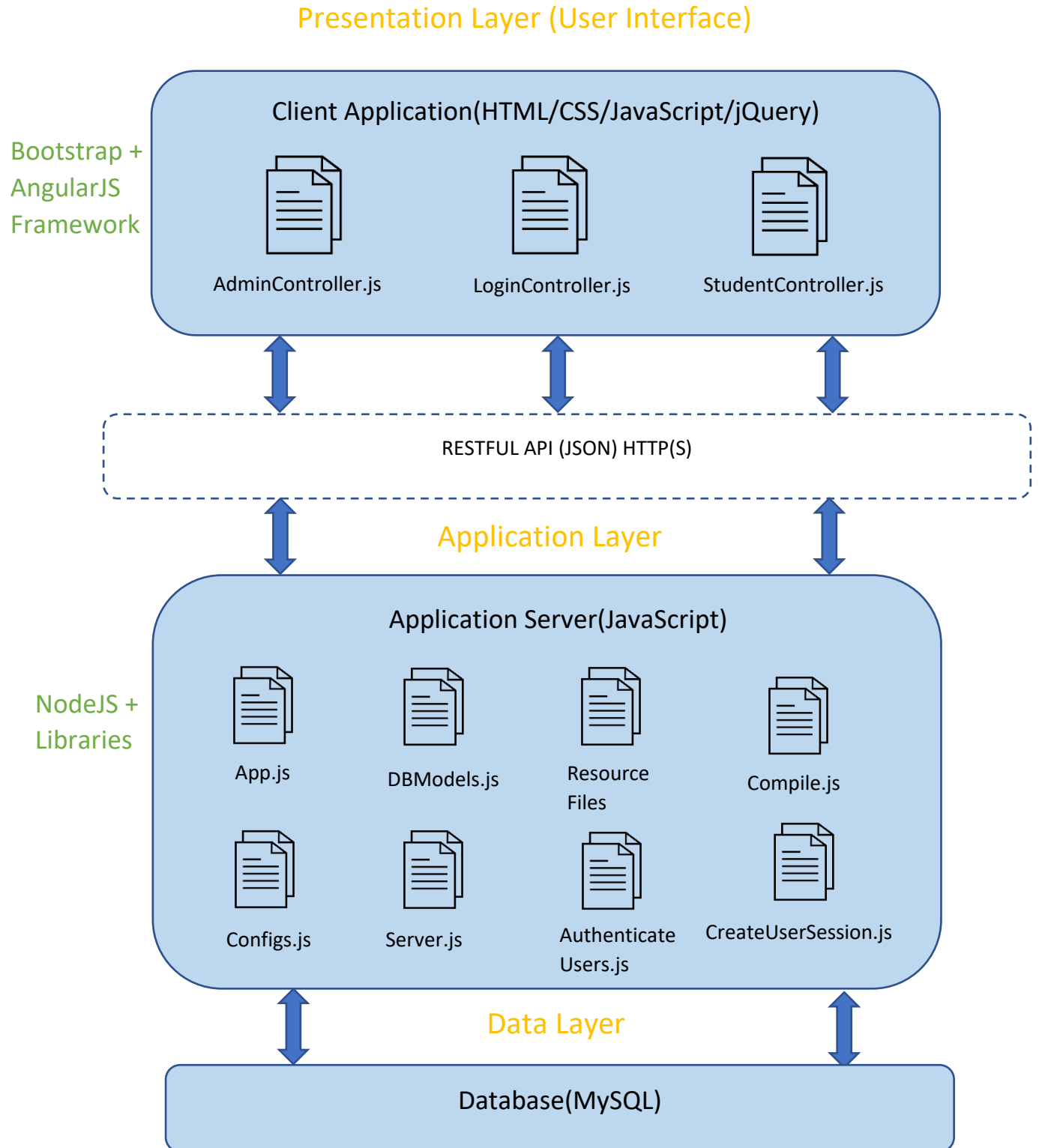


Figure 3-1. Overview of the system architecture



### 3.3.1 RESTful Client-Server Architecture

REST (REpresentational State Transfer) is a style of web architecture that is chosen for this project. Data are sent to the application server using JSON. The REST API has the following characteristic:

**Stateless client/server protocol:** Both the client and the server does not need to need to remember any of the previous states. If data is required, it must be sent along with the request. Typical data transaction in a RESTful architecture takes place using POST, GET, PUT and DELETE. However, in this project, only POST and GET are used. REST also caches server responses in the client itself, so that a client does not need to make a server request for the same resource repeatedly.

An advantage of the REST API is that it can support multiple languages, in our case, AngularJS and JavaScript. This would mean that REST API is scalable even if there are more frameworks. Furthermore, as the application is using Node.js, parsing JSON data will be very convenient.

### 3.3.2 Presentation Layer

The Presentation Layer refers to the User Interface(UI) of this system where users can perform their tasks via graphic elements. The display of the UI frontend is written using HTML and CSS while the controller is coded in JavaScript and jQuery. The UI utilizes Bootstrap and AngularJS Framework to provide a responsive website.

## **Bootstrap**

Bootstrap was chosen for this web application because of the following reasons:

### **Improving the speed of development**

As this project is time intensive, it is crucial for time to be spent on the right development areas. Bootstrap provides readymade components like forms, buttons, and tables and it is easy for these codes to be modified if needed to suit the purpose of this project. In addition, Bootstrap also contains a wide range of JavaScript Libraries to be leveraged on.

### **Cross Browser Compatibility**

One of the assumptions is that users will be using the system through different browsers, thus it is important that the interface must be consistent and compatible across all these browsers.

### **Future Development and Maintenance**

Bootstrap has a vast community that is constantly trying to improve the framework. Bootstrap also has a comprehensive amount of documentation, making it easier for future developers who are trying to maintain or add new features to this system.

## **AngularJS**

AngularJS framework is used along Bootstrap in the Presentation Layer. JAAS leverages on some of the advantages that AngularJS provides, such as dynamic data-binding, control over DOM (Document Object Model) structure, form validation as well as customizing components and then reuse it.

In this project, data binding is used largely to bind data from the server to variables declared in the client side. JAAS also make use of the AngularJS's ng-repeat directive to create a table with dynamic values from values in the database.

SCE17-0389  
Java Program Auto Assessment

Students 10 records per page Search:

Student ID	Student Name	Lab Group	Email
1	student	Lab04	JHUANG035@e.ntu.edu.sg
8	john	123	jianwei1993@hotmail.com
11	Tim	12345	gfg22f@hotmail.com

Showing 1 to 3 of 3 entries ← Previous 1 Next →

Figure 3-2. using ng-repeat to dynamically create a table from database values

In short, AngularJS provides a basis for building a CRUD (Create, Read, Update, Delete) web application.

### 3.3.3 Application Layer

The Application Layer is the central layer that deals with the logic of the web application. It receives data from the Presentation Layer and processes these data. The application Layer also interacts with the Data Layer directly to perform CRUD operations. In this project, the logic that performs the assessment on the student's code or program lies in this layer. Express was chosen to be used for this layer.

#### Express.js

Express is a web-application framework built on top of Node.js, inheriting the existing advantages of Node.js. Node.js is event-driven and non-blocking, hence providing an improved performance over several non-asynchronous frameworks.

JavaScript is used as the language for Node.js, reducing the need for an extra language for the web application stack. As both the client and server are both using the same language to transfer data, there are fewer complications for errors in the process.

The Node Package Module(NPM) also provides a simple and convenient way to download the many available libraries and packages. Utilizing these libraries is crucial to the development of this web application as it is not feasible to develop certain functions within the timeframe of this project.

Some of the notable libraries are as followed:

**Node-cmd (source: <https://www.npmjs.com/package/node-cmd>)**

The Node-cmd is a command line/terminal interface used in this project to perform simple file operations like move, copy, and deletion of files.

```
var cmd=require('node-cmd');

cmd.get(
  'javac Labs/'+req.body.username+'/*.java',
  function(err, data, stderr){
    if (err) {
      flag = 1;
    }
  });
```

*Figure 3-3. Node-cmd is used to compile Java files*

**Exceljs (source: <https://github.com/guyonroche/exceljs>)**

Exceljs is a library used to perform operations such as writing and reading data to and from Excel spreadsheets. This library is used specifically to allow admin to upload a list of students from an excel sheet directly to the database.

**Node-datetime (source: <https://www.npmjs.com/package/node-datetime>)**

Node-datetime is used to track the date and time of submission for student's lab solutions and assessment questions.

**Extract-zip (source: <https://www.npmjs.com/package/extract-zip>)**

In the case where multiple files are required for submission, e.g. students are required to zip up their files to be submitted. Once uploaded to the server, these Java files need to be extracted out so that they can be compiled and run.

**Pdfkit (source: [http://pdfkit.org/docs/getting\\_started.html](http://pdfkit.org/docs/getting_started.html))**

To generate reports for students after they submitted their codes or programs, the pdfkit library is used. Besides writing text to the pdf, this library also allows images to be placed in the generated pdf.

**Mysql (source: <https://github.com/mysqljs/mysql>)**

Mysql is a node.js driver for MySQL. By specifying the host, user, and password, a connection can be made to the database, creating a tunnel for CRUD operations.

**Child\_process (source: [https://nodejs.org/api/child\\_process.html](https://nodejs.org/api/child_process.html))**

The most important module for JAAS. Child\_process allows the server to spawn an instance of the student's program after compilation. By having a reference to this child\_process, the server is able to send input to the student's program and test for the program's expected output. Thus, by attaching a listener to the child process, the server is able to check if the output of the program is correct.

### 3.3.4 Data Layer

The Data Layer resides at the bottom of the 3-tiered architecture. In this project, both MySQL and plain text files are used as data storage. JAAS make use of the Mysql library in the application layer to perform queries on the actual database.

#### MySQL

MySQL is chosen for the Database Management System(DBMS) of this web application due to its ease of use, data security, scalability as well as being an open source software.

## 3.4 Database Design

The section below shows the configurations, tables, and relationships between the entity for this project.

### 3.4.1 Database Configuration

Establishing a connection to MySQL database using the Mysql interface provided by Node.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "93829359",
  database : 'fyptable'
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});

module.exports = con;
```

Figure 3-4. Configuration of the database in config.js

### 3.4.2 Database Entity Relationship Diagram

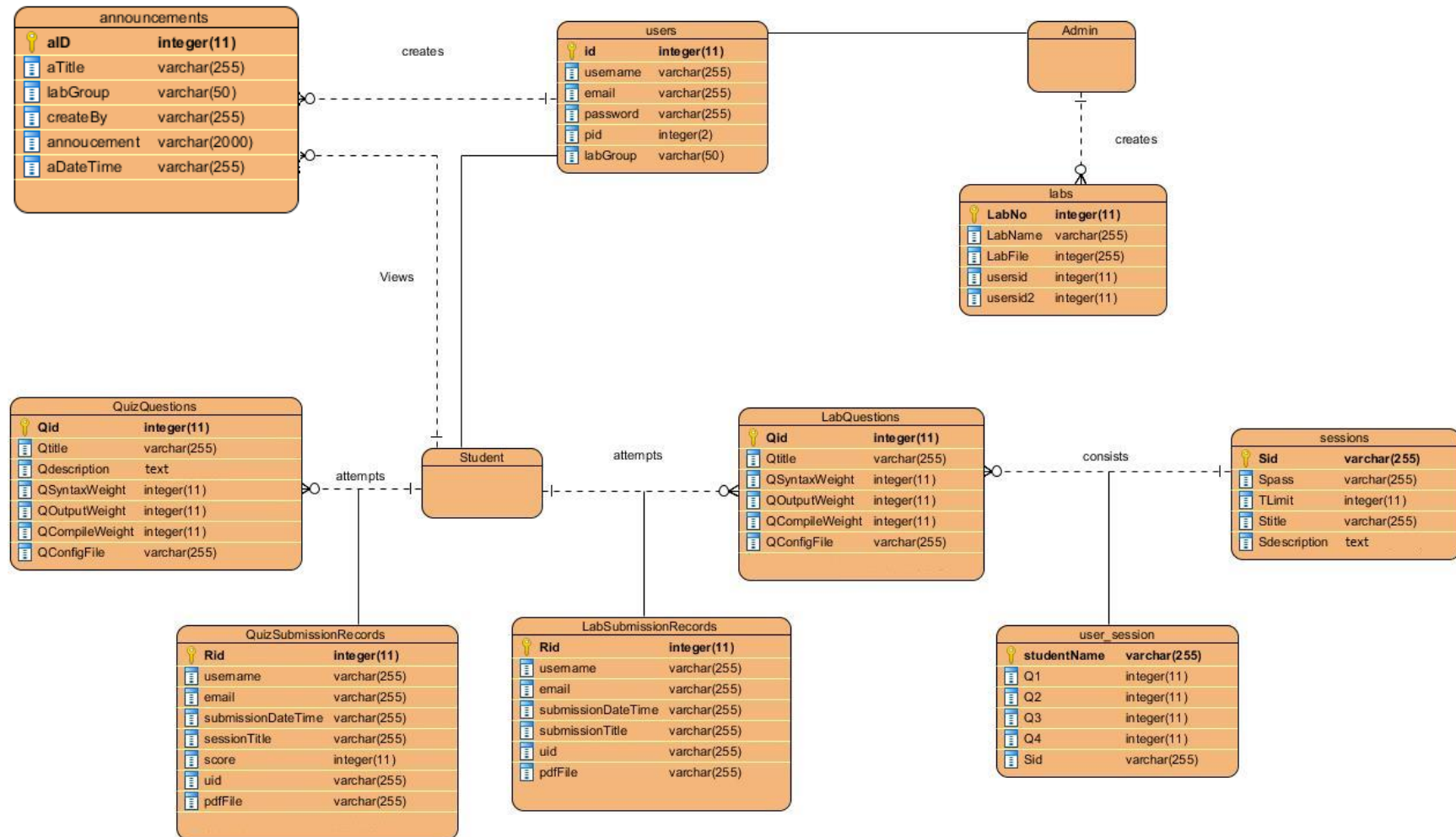


Figure 3-5. Entity Relationship Diagram of JAAS

### 3.4.3 Database Tables

**Table: users**

Field	Type	Null	Key	Extra
id	int(11)	No	PRI	auto_increment
username	varchar(255)	No	-	-
email	varchar(255)	Yes	UNI	-
password	varchar(255)	No	-	-
pid	int(2)	No	-	-
labGroup	varchar(50)	Yes	-	-

*Figure 3-6. User table*

**Description:** The user table stores the information of every user in the system which is used to authenticate users. Information for each user is recorded during each lab assignment and assessment question submissions.

**Attributes:**

**id:** Primary Key of the users table.

**username:** Actual name of the user. Used to authenticate a user into the system along with the password field. As there can be multiple users with the same name, it is not used as Primary Key.

**email:** The unique email of each student. Used to uniquely identify each user as there cannot be duplicate emails.

**password:** password used to authenticate users into the system in conjunction with a username.

**labGroup:** used to categorize student into different lab groups.



**Table: labs**

Field	Type	Null	Key	Extra
LabNo	int(11)	No	PRI	-
LabName	varchar(255)	No	-	-
LabFile	varchar(255)	No	UNI	-

*Figure 3-7. labs table*

**Description:** The labs table store the basic information of each labs which will be used to display to the students.

**Attributes:**

**LabNo:** Used to identify each lab

**LabName:** Table of the labs

**LabFile:** File name of the lab pdf file. E.g. lab1.pdf.

**Table: announcements**

Field	Type	Null	Key	Extra
aID	int(11)	No	PRI	auto_increment
aTitle	varchar(255)	No	-	-
labGroup	varchar(50)	Yes	-	-
createBy	varchar(255)	No	-	-
announcement	text	No	-	-
aDateTime	varchar(255)	No	-	-

*Figure 3-8. announcements table*

**Description:** The announcements table stores the information of announcement created by admin.

**Attributes:**

**aID:** The Primary Key of the data and is used to uniquely identify each announcement.

**aTitle:** Title of the announcement

**labGroup:** use to check if the announcement should be displayed for a specific student user

**createBy:** The creator of the announcement

**announcement:** The main body of the announcement, also the description

**Table: LabSubmissionRecords**

Field	Type	Null	Key	Extra
Rid	int(11)	No	PRI	auto_increment
username	varchar(255)	No	-	-
email	varchar(255)	No	-	-
submissionDateTime	varchar(255)	No	-	-
submissionTitle	text	No	-	-
score	int(11)	No	-	-
uid	varchar(255)	No	-	-
pdfFile	varchar(255)	No	-	-

*Figure 3-9. Labsubmissionrecords table*

**Description:** The LabSubmissionRecords table stores and track the information of each submission by students such as title, date, time. Students and admin can then view past attempts of these lab submission records.

**Attributes:**

**rid:** Primary Key of the LabSubmissionRecords Table. Since there could be multiple submission by the same user, the rid is used to distinguish between different submission.

**username:** Author of the lab submission

**email:** email of the Author who submitted the lab assignment

**submissionDateTime:** Date and time of the submission. Can be used to determine if student submit their assignment late.

**submissionTitle:** Title of the lab submission. Used to compare against the questions in the Lab questions table

**score:** The total score that was computed by adding up the compilation/syntax score and output score.

**uid:** a random number that is generated using current date time to differentiate between different submission.

**pdfFile:** The name of the pdfFile that was generated for the specific lab submission. Uid is appended to the name of the pdfFile so that a new copy will be created instead of being overwritten.

**Table: QuizSubmissionRecords**

Field	Type	Null	Key	Extra
Rid	int(11)	No	PRI	auto_increment
username	varchar(255)	No	-	-
email	varchar(255)	No	-	-
submissionDateTime	varchar(255)	No	-	-
sessionTitle	text	No	-	-
score	int(11)	No	-	-
uid	varchar(255)	No	-	-
pdfFile	varchar(255)	No	-	-

*Figure 3-10. quizsubmissionrecords table*

**Description:** Similar to the LabSubmissionRecords table, the QuizSubmissionRecords table stores and track the information of each submission by students such as title, date, time. Students and admin can then view past attempts of these quiz submission records.

**Attributes:**

**Rid:** Primary Key of the QuizSubmissionRecords Table.

**username:** Author of the quiz submission

**email:** email of the Author who submitted the solutions for the quiz session.

**submissionDateTime:** This attribute is used to track the date and time of the submission

**sessionTitle:** Title of the session submission.

**score:** The total score that was computed by adding up the compilation/syntax score and output score.

**uid:** a random number that is generated using current date time to differentiate between different submission.

**pdfFile:** The name of the pdfFile that was generated for the specific quiz submission. Uid is appended to the name of the pdfFile so that a new copy will be created instead of being overwritten.

**Table: sessions**

Field	Type	Null	Key	Extra
Sid	varchar(255)	No	PRI	-
Spass	varchar(255)	Yes	-	-
TLimit	Int(11)	Yes	-	-
Stitle	varchar(255)	Yes	-	-
Sdescription	text	Yes	-	-

*Figure 3-11. sessions table*

**Description:** The sessions table stores the basic information of each quiz session. It is used to authenticate a user for the session.

**Attributes:**

**Sid:** Session id that is used with Spass to authenticate a student user into a quiz session.

**Spass:** Session password to access a specific quiz session

**TLimit:** Time limit of the quiz session

**Stitle:** Title of the quiz session

**Sdescription:** Description of the quiz session

**Table: user\_session**

Field	Type	Null	Key	Extra
studentName	varchar(255)	No	PRI	-
Q1 to Q4	int(11)	No	-	-
Sid	varchar(255)	No	PRI	-

*Figure 3-12. user\_sessions table*

**Description:** Each row in the user\_session table binds a user to a session containing a set of questions.

**Attributes:**

**studentName:** student user who is the owner of a specific session.

**Q1-Q4:** Random set of questions generated from the quizQuestions table.

**Sid:** Primary Key of the session table. Use to tie a specific session to a student.

**Table: QuizQuestions**

Field	Type	Null	Key	Extra
Qid	Int(11)	No	PRI	auto_increment
Qtitle	varchar(255)	No	-	-
Qdescription	text	No	-	-
QSyntaxWeight	Int(3)	No	-	-
QOutputWeight	Int(3)	No	-	-
QCompileWeight	Int(3)	No	-	-
QconfigFile	varchar(255)	No	-	-

*Figure 3-13. quizQuestions table*

**Description:** The quizQuestion table stores the simple information of each question which is used to display to the students during a quiz session

**Attributes:**

**Qid:** Unique identifier for each question. As Question title can be duplicated, Qid is used as the primary key.

**Qtitle:** Question Title to be displayed.

**Qdescription:** Description of each question to be displayed during a session

**QSyntaxWeight:** Determine how much syntax score is factored into the calculation of the total score

**QOutputWeight:** Determine how much output score is factored into the calculation of the total score

**QCompileWeight:** Determine how much score should a successful compilation be given.

**QconfigFile:** filename which inputs, expected output, keywords and keyword score are specified.

**Table: LabQuestions**

Field	Type	Null	Key	Extra
Qid	Int(11)	No	PRI	auto_increment
Qtitle	varchar(255)	No	PRI	-
QSyntaxWeight	Int(3)	No	-	-
QOutputWeight	Int(3)	No	-	-
QCompileWeight	Int(3)	No	-	-
QconfigFile	varchar(255)	No	-	-

*Figure 3-14. LabQuestions table*

**Description:** The labQuestion table stores the values which are used to computed student's submission for the labs. Data will not be pulled out from this table to be displayed.

**Attributes:**

**Qid:** Unique identifier for each question, Qid is used as the primary key.

**Qtitle:** Question title that identifies each unique lab question

**QSyntaxWeight:** Determine how much syntax score is factored into the calculation of the total score

**QOutputWeight:** Determine how much output score is factored into the calculation of the total score

**QCompileWeight:** Determine how much score should a successful compilation be given.

QconfigFile: filename which inputs, expected output, keywords and keyword score are specified.

### 3.5 Summary of Tools and Technologies used

Web Application Stack	Name of tool/technology
Front-End/User Interface Framework	AngularJS
Web Application Framework	Express.js
Client-Side Scripting Language	JavaScript
Server-Side Scripting Language	JavaScript
Database	MySQL 5.7
Integration Development Environment	Eclipse, Sublime Text 3, Notepad++

*Figure 3-15. Summary of Tools and Technologies used*

## 4. Implementation

This section of the report will cover the implementation of each function stated in the use case.

The system overview is depicted by the conceptual class diagram shown in figure 4-1. It illustrates the relationship between the different classes.

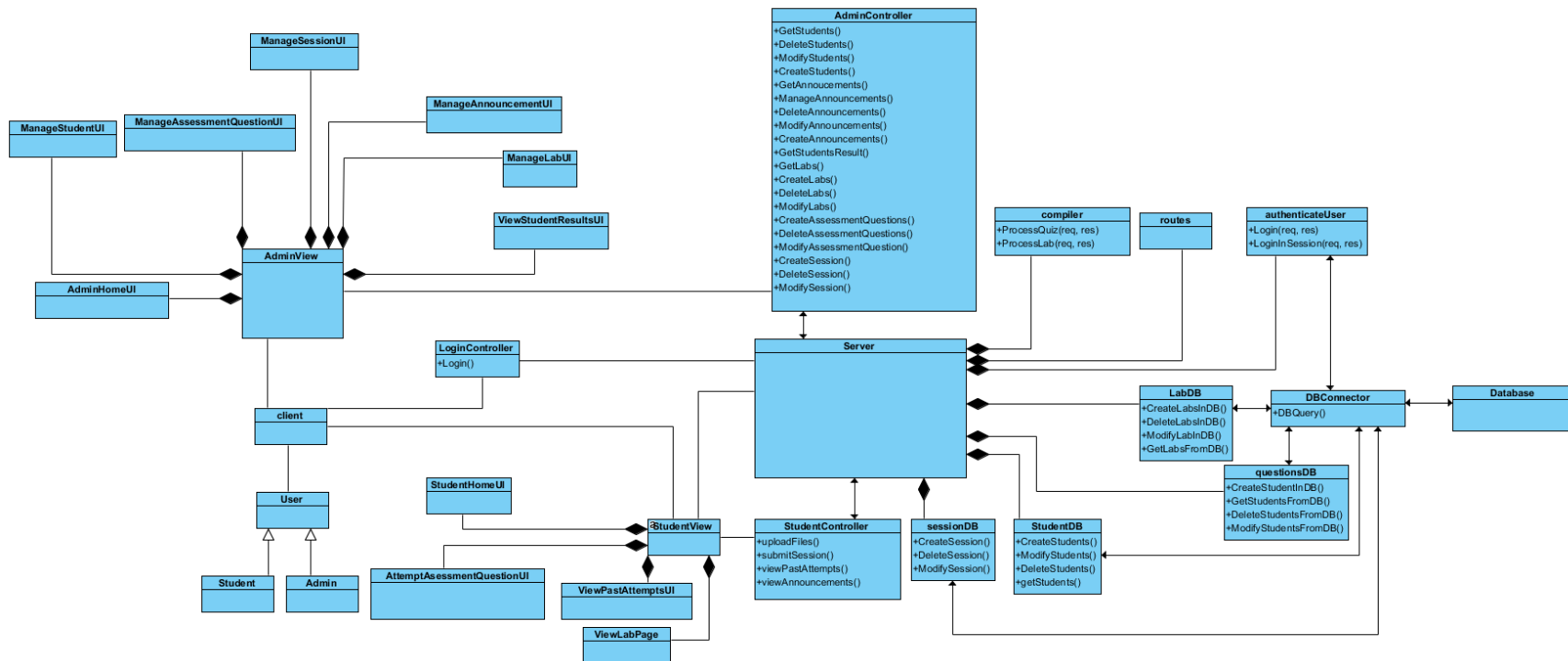


Figure 4-1. Conceptual Class Diagram

## 4.1 System Functions

Several scenarios of interaction between the user and the system which are more complex, are shown in the following sequence diagrams to provide a better understanding of what will happen during execution of a use case.

### User- Login

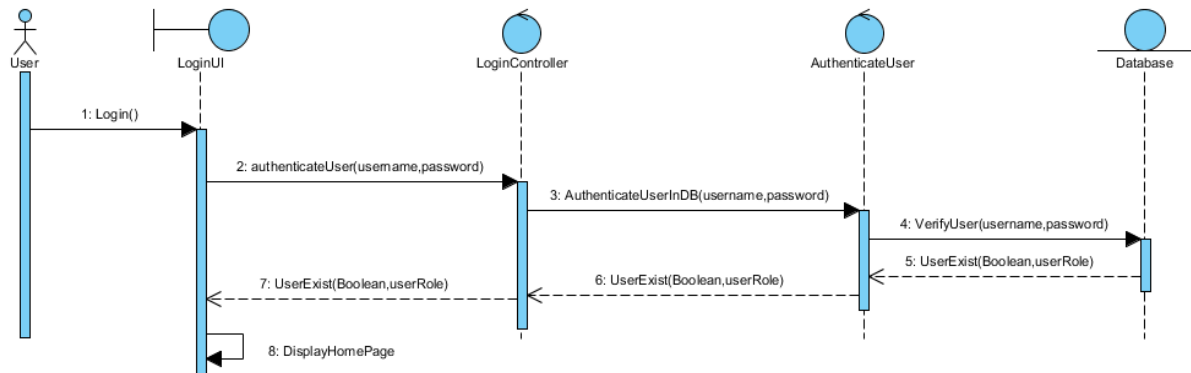


Figure 4-2. Sequence Diagram: User Login

### Student - Lab Submission

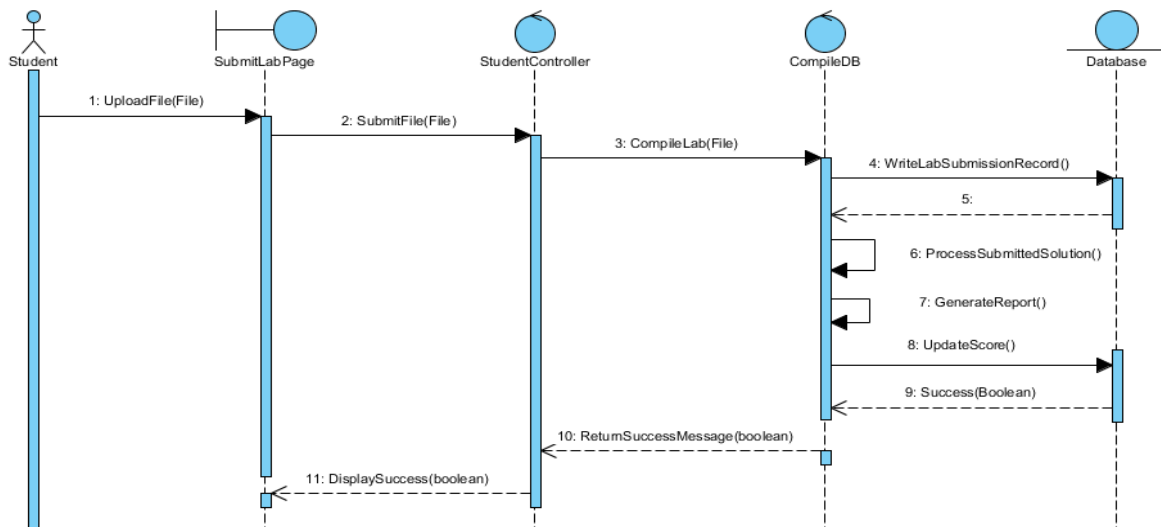


Figure 4-3. Sequence Diagram: Lab Submission



## Student – Quiz Session Submission

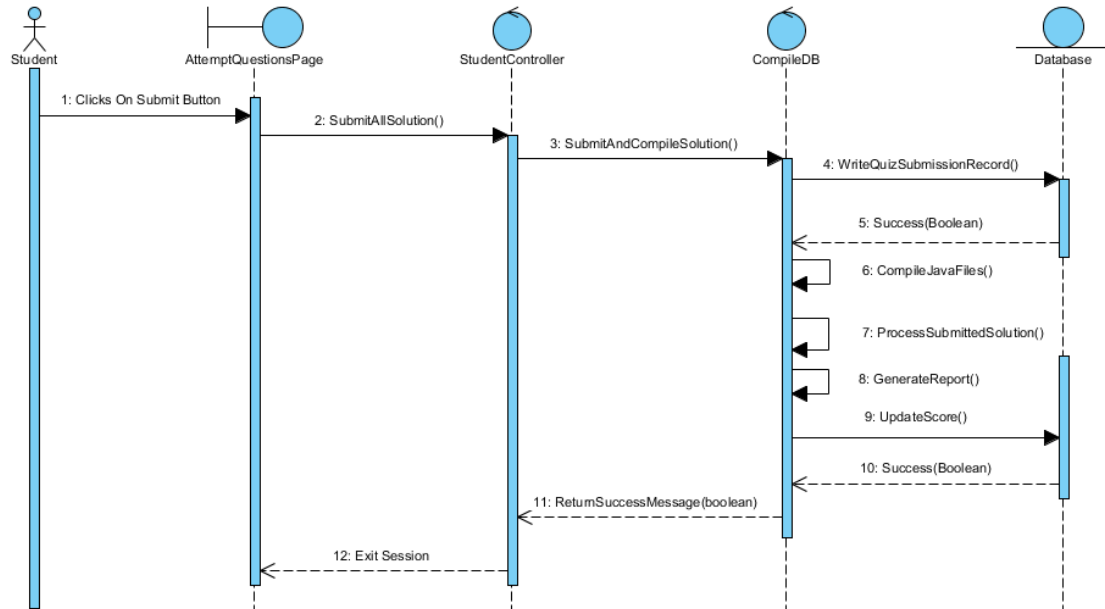


Figure 4-4. Sequence Diagram: Quiz Submission

## Admin – Create Assessment Question

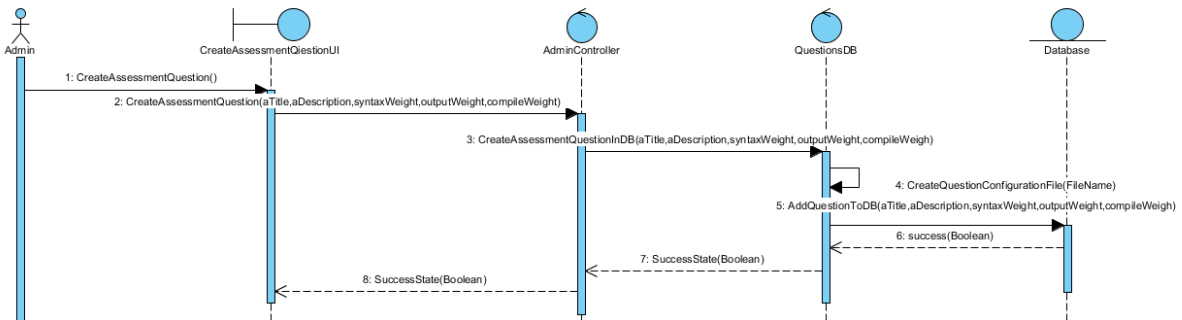


Figure 4-5. Sequence Diagram: Create Assessment Question

## Admin – Delete Students

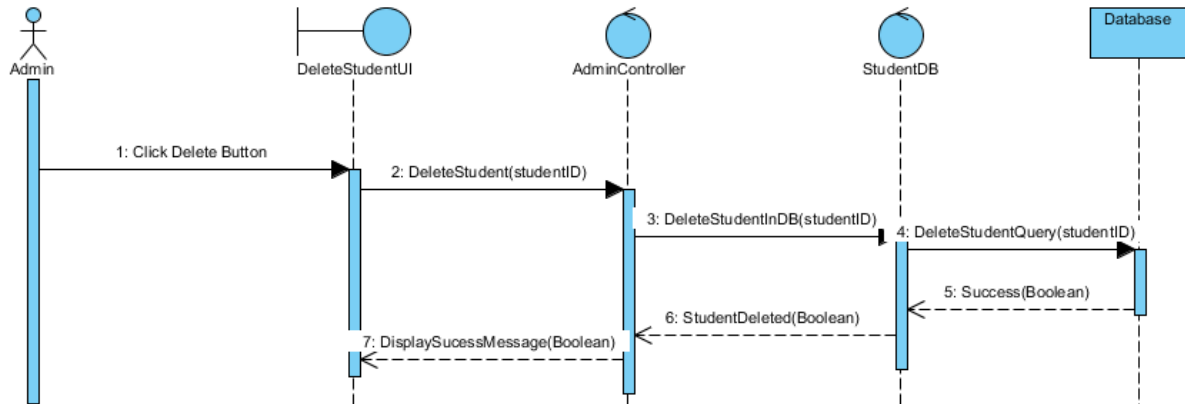


Figure 3-6. Sequence Diagram- Delete Students

## Admin – View Students' Attempts

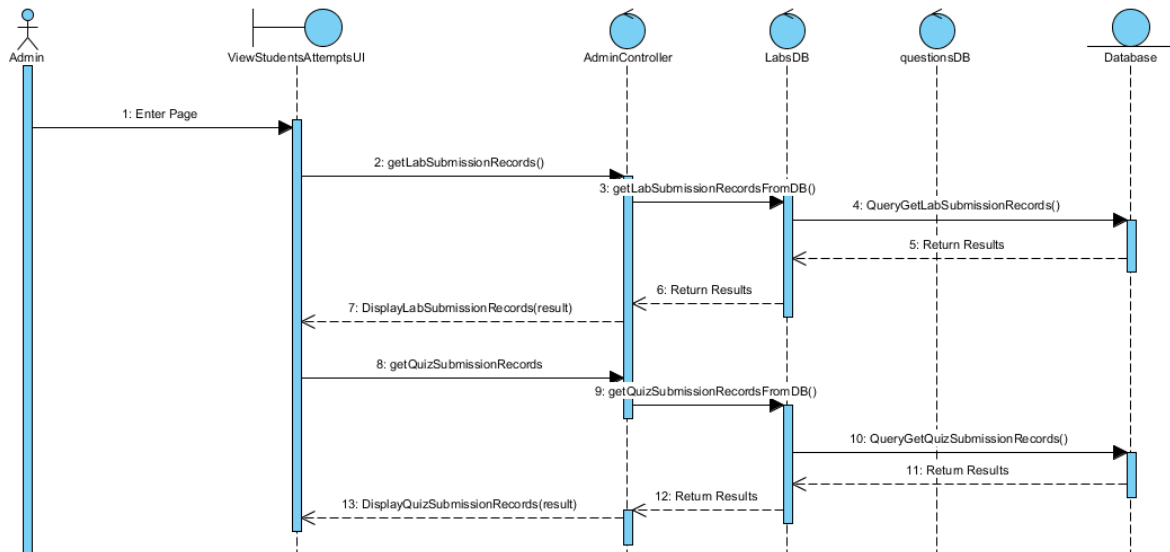


Figure 4-7. Sequence Diagram- View Students' Attempts

### 4.1.1 Login Page

Upon entering localhost://3000 the user will be shown a login page. Users are required to be authenticated into the system by entering their credentials. The credentials will then be POST to the server which will run a query on the user table to check if the credential is valid. If the correct username and password combination is entered, the user will be directed to the homepage of their respective role.

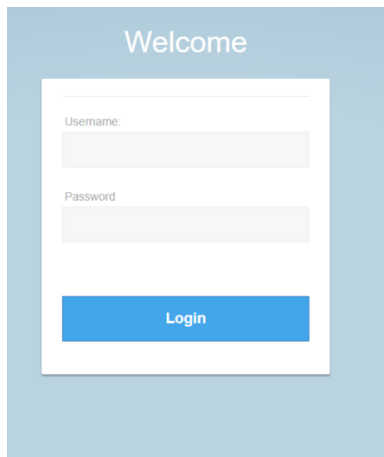


Figure 4-8 Successful Login

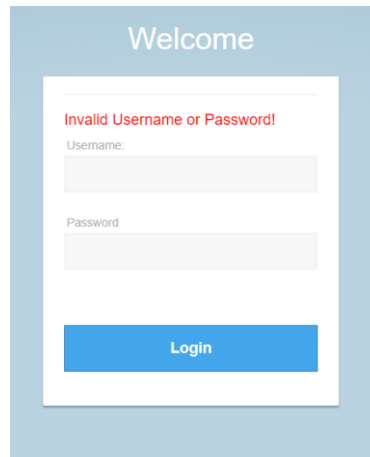


Figure 4-9 Unsuccessful login

## 4.2 Student Function Implementation

### 4.2.1 Student Home Page

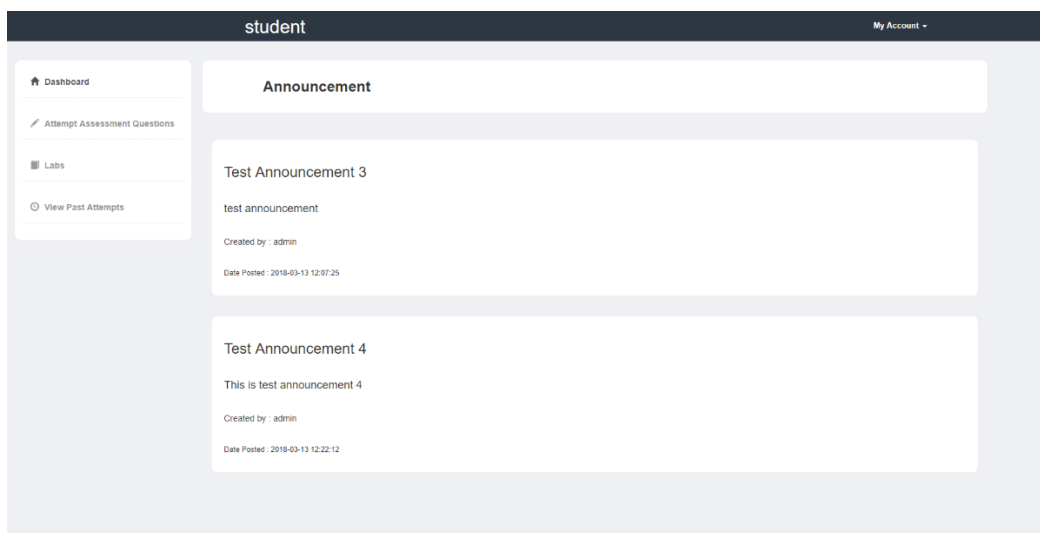


Figure 4-10 Student Home page displaying announcements

The student home page will display all announcement that belongs to the lab group which the student is categorized under. The announcement title, announcement description, the creator of the announcement creator will be listed.

```
<div data-ng-init="getAnnouncements()">

<div ng-repeat="announcement in announcements">
<div class="content-box-large">

<div>
  <h3>{{announcement.aTitle}} </h3><br>
  <h4>{{announcement.announcement}}</h4> <br>
  <h5> Created by : {{announcement.createBy}} </h5> <br>
  <h6> Date Posted : {{announcement.aDateTime}} </h6>
</div>
</div>
</div>
</div>
```

Figure 4-11 Displaying announcements using AngularJS Binding

Upon landing on the home page, the AngularJS directive “data-ng-init” will call a client-side function called getAnnouncement() which will perform a POST to the route.js file and call a server-side function which will then query the database and retrieve all announcement that belongs to the student. The results are returned via JSON back to the client-side function.

```
$scope.getAnnouncements = function(){

$http.post('/getStudentAnnouncements',{data:{ email : $scope.email }}).then(function(res){

    $scope.announcements = res.data;

}).catch(function(error){

    });

}
```

Figure 4-12. POST request to the server retrieve announcements from the database

```
exports.getStudentAnnouncement = function(req,res){
db.query('SELECT * FROM users WHERE email= ?', [req.body.data.email], function (error, results, fields){

if(error){
console.log('Code 400, Error occurred');
console.log(error);
res.json({ state: 0 });
}

else{
db.query('SELECT * FROM announcements WHERE labGroup= ?', [results[0].labGroup], function (error, results, fields){
if(results.length > 0 ){
res.json(results);
}
});
}
});
}
```

Figure 4-13. Server-side Query to retrieve announcements to the student

## 4.2.2 Attempt Assessment Questions – Log in Session

student

My Account

Dashboard

Attempt Assessment Questions

Labs

View Past Attempts

Log on to Assessment Session

Session ID:

Session Password:

Enter

Figure 4-14. Login to Assessment Quiz Session

The assessment question tab will direct a student user to the login page of the assessment session. A unique Session ID defines a session. Valid sessions are stored in the database and if a correct Session ID and Session password are entered, the session's title, description, and a button to start the assessment session will be displayed.

student

Dashboard

Attempt Assessment Questions

Labs

View Past Attempts

Test Assessment Session

Time Limit : 90 minutes

This is a test session. This session consist of 4 questions weighing 25 marks each. Please try to attempt all questions. Click the button below to start your attempt. Maximum of 1 attempt(s).

Start

Figure 4-15. Session Title and description are displayed

If the user has not previously attempted the session, the server will generate a set of questions from a pool using the algorithm shown below.

```
exports.createNewSession = function(req,res){
//Select the Qid column from Quiz Sessions.
db.query('SELECT Qid FROM QuizQuestions', function (error, results, fields){

var sid = req.body.data.sessionID;
var error =0;
var dupe =[];
var arr = [];
var i =0;
var flag = 0;
var proceed =0;
var duplicate =0;

//To generate 4 unique questions
while(i < 4){
//flag will only be set to 1 when a question number has successfully stored into the Question Array
while(flag==0){ // While random generated number has duplicate
var randomnumber = Math.floor(Math.random()*results.length); //Generate a random number using the no. of questions in the Database
//loop through array and check for duplicates
for(var u=0; u < dupe.length; u++){
if(randomnumber == dupe[u]){
duplicate++; //if duplicate found, increment duplicate flag.
}
}

if(duplicate > 0){
flag = 0;
duplicate =0;
}
//if there is no duplicate, store question into result set.
else{
flag = 1;
duplicate = 0;
arr[i] = results[randomnumber];
dupe[i] = randomnumber;
}
}
i++;
flag =0;
}
}
```

Figure 4-16. Code of function which will generate pick 4 unique questions from a pool of x questions

To summarize, this algorithm works by fetching all **Qid** from QuizQuestions database and generating a value using `math.random()` based the count of the Qid. The random value between 0 to count(Qid) will then be used to fetch the specific Qid from the initial query and put it into the result set, if it does not already exist in the result set.

Step 1.) Query fetches all rows of questions in Quiz Database (e.g. 12 rows)

Step 2.) `var randomNumber = Math.floor(Math.random()*12)` (e.g. 6 is generated)

Step 3.) check if dupe set contains 6, if it does, set the duplicate flag to true else do nothing

Step 4.) check if the duplicate flag is true. If true, redo Step 2. else add `arrayOfQid[6]` (e.g. 6) into result set.

Step 5.) Repeat Step 2 to 4 until the result set satisfies the required amount of questions for the session.

Step 6.) The final result set will contain 4 unique value e.g. [ 2, 5, 4, 1]

Step 7.) Insert user, session, and the generated question to the `user_sessions` table

```
mysql> select * from user_session;
```

studentName	Q1	Q2	Q3	Q4	Sid
JHUANG035@e.ntu.edu.sg	3	2	8	7	Lab04
JHUANG035@e.ntu.edu.sg	7	3	6	8	Quiz01
jimmy@gmail.com	3	5	8	6	Quiz01

Figure 4-17. user\_sessions table that binds a user to a quiz session

Upon entering the Quiz Assessment Session, the controller will use AngularJS data-ng-init and call getQ1-Q4 functions to retrieve titles and descriptions to be displayed on the page.

## Client-Side

```
<div class="row">
<!-- to display content here -->

<div class="content-box-large" data-ng-init="getQ1()">
<div class="container">

<br>
<h3>Question 1.</h3>
<br>
<br>
<h3>
({{Q1Title}})</h3>
</br>
<br>
<div style="font-size:14px;width:800px;">
({{Q1Description}})
</div>
<br>
<div id="editorContainer">
<div id="editor1"></div>
<br>

<b>Output:</b>
<textarea class="form-control" rows="4" name="Q1Output" id="Q1Output" ng-model="Q1Output" style="width:800px;" disabled></textarea>
<br>
<button id="runQ1Btn" class="btn btn-success btn-lg" type="submit" ng-click="compileQ1()" style="float:right;">Save and Compile</button>

</div>
</div>
</div>
</div>

$scope.getQ1= function(){
$http.post('/getQ1',{data:{ sessionUser: $scope.username, sessionID: $scope.sessionSid, email : $scope.email }}).then(function(res){
$scope.Q1Title = res.data.q1Title;
$scope.Q1Description = res.data.q1Description;
}).catch(function(error){
}):
})
```

Figure 4-18 Client-Side codes to request for Question's details to be displayed

## Server-Side

```
exports.getQ1fromDB = function(req,res){
var email = req.body.data.email;
var questionSession = req.body.data.sessionID;
var q1Question =0;

db.query('SELECT Q1 FROM user_session WHERE Sid= ? AND studentName=?',[questionSession,email], function (error, results, fields){
if(error){
console.log(error);
}
else{
q1Question = results[0].Q1;

db.query('SELECT Qtitle,Qdescription FROM QuizQuestions WHERE Qid= ?', [q1Question], function (error, results, fields){
if(error){
console.log(error);
}
else{
res.json({ q1Title : results[0].Qtitle, q1Description : results[0].Qdescription});
}
}):
});
});}
```

Figure 4-19 Retrieving question title and question description from QuizQuestions Database

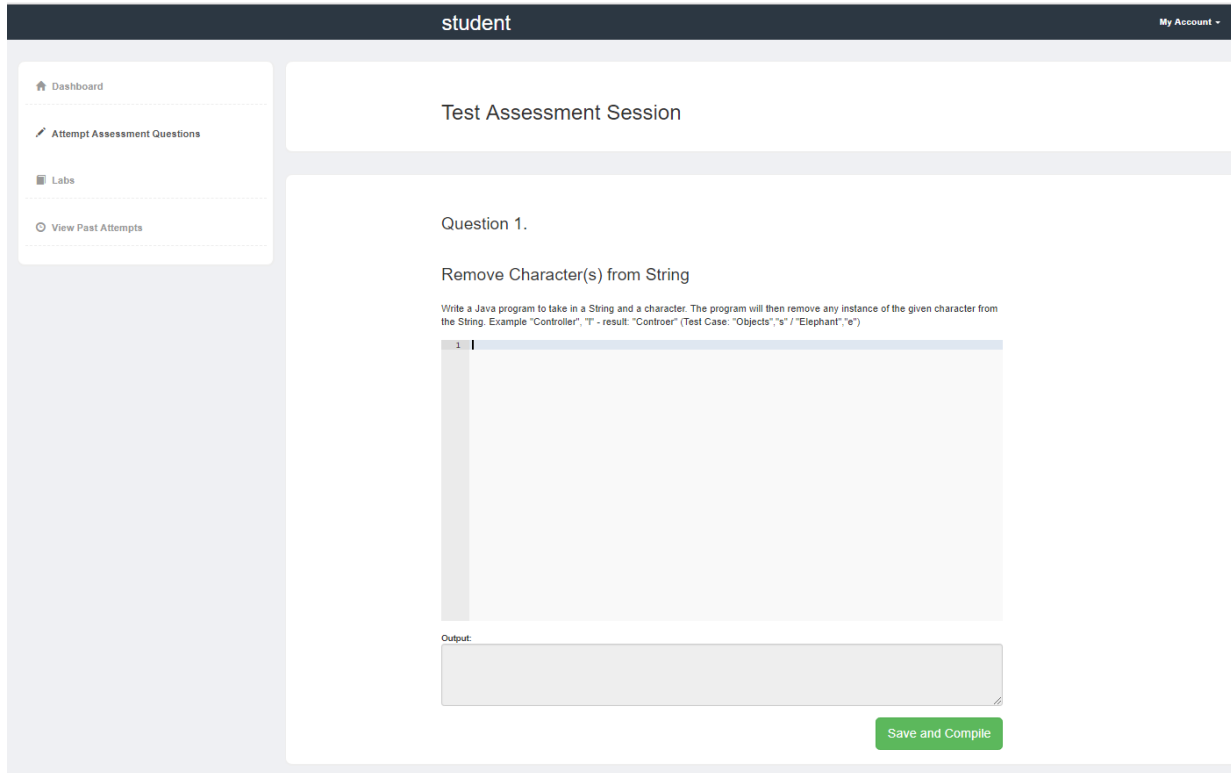


Figure 4-20 Displaying the Assessment Questions Page

### 4.2.3 Attempt Assessment Questions – Save and Compile Question

Ace editor(<https://ace.c9.io/>) is used as an alternative to standard textbox display as it has a line count display, making it easier for students to copy their code from their IDE and paste in this platform.

For each question, there will be a button that enable the students to submit their codes to be compiled at the server side. If there is no error in compilation, the output field will show a “Compile Success” message and a class file will be generated as shown in Figure 4-21. If the codes cannot be compiled due to a syntax error, an error message will be shown in the output field.



SCE17-0389  
Java Program Auto Assessment

studentMy Account

Dashboard

Attempt Assessment Questions

Labs

View Past Attempts

Test Assessment Session

Question 1.

String Reversal

Write a Java program to take in a String and reverse it using an iterative or recursive method. (Example "Tank" - "knaT", "Table" - "elbaT")

```
1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 public class Q1 {
5
6     public static void main(String[] args) {
7         Scanner in = new Scanner(System.in);
8         System.out.println("Enter a String:");
9         String input = in.nextLine();
10
11         System.out.println("Result : ");
12
13         for(int i=input.length()-1;i>=0;i--){
14             System.out.print(input.charAt(i));
15         }
16
17     }
18 }
19
20
21
```

Output:  
Compile Success!

Save and Compile

PC > New Volume (D:) > Java-Assessment > Java-Assessment > Java-Assessment-Project > Quiz > student

Name	Date modified	Type	Size
Q1.class	3/14/2018 2:27 AM	CLASS File	1 KB
Q1.java	3/14/2018 2:27 AM	JAVA File	1 KB

Figure 4-21 .java and .class file generated after student submits code.

The server-side first uses the ‘fs(filesystem)’ package to write the student’s submitted code into a .java file and run a cmd command to compile the .java file into a class file. E.g. “javac Quiz/studentName/Q1.java”. If successful, the server will send a response to the client saying the code has successfully been compiled. If there is an error, the reason for the fail compilation will be sent back to the client.

## SCE17-0389

### Java Program Auto Assessment

```
app.post('/compileQ1',function(req,res){
  var Q1Title = req.body.data.questionTitle;
  var Q1code = req.body.data.Q1code;
  var studentName = req.body.data.studentName;

  if (!fs.existsSync(__dirname + '/../Quiz/'+req.body.data.studentName)){
    fs.mkdirSync(__dirname + '/../Quiz/'+req.body.data.studentName);
  }

  fs.writeFile(__dirname + '/../Quiz/'+req.body.data.studentName + '/Q1.java', Q1code, function(err) {
    if(err) {
      return console.log(err);
    }

    else{
      console.log("The file was saved!");
    }

    output = cmd.get(
      'javac Quiz/'+studentName+'/Q1.java',
      function(err, data, stderr){
        if(err){
          console.log(err);
          res.json({ state : 0, error : err.toString()});
        }
        else {
          res.json({ state : 1});
        }
      });
    }
  });
});
```

Figure 4-22. Server-side code to write the java file and compile the corresponding class file

student My Account

Dashboard

Attempt Assessment Questions

Labs

View Past Attempts

### Test Assessment Session

Question 1.

#### String Reversal

Write a Java program to take in a String and reverse it using an iterative or recursive method. (Example "TanK" -> "KaNt", "Table" -> "elbat")

```
1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 public class Q1 {
5
6     public static void main(String[] args) {
7         Scanner in = new Scanner(System.in);
8         System.out.println("Enter a String:");
9         String input = in.nextLine();
10        System.out.println("Result : ");
11        for(int i=(input.length()-1);i>=0;i--){
12            System.out.print(input.charAt(i));
13        }
14    }
15 }
16
17
18
19
20
21
```

Output:

```
Error: Command failed: javac Quiz/student/Q1.java
Quiz/student/Q1.java:7: error: ';' expected
Scanner in = new Scanner(System.in);
^
```

Save and Compile

Figure 4-23. An unsuccessful attempt to compile a file due to syntax error (missing semi-colon in this case)

## 4.2.4 Attempt Assessment Questions – Save and Submit Solution

localhost:3000/questionPage

### Question 4.

#### Remove Character(s) from String

Write a Java program to take in a String and a character. The program will then remove any instance of the given character from the String. Example "Controller", "r" - result: "Controlei" (Test Case: "Objects", "s" / "Elephant", "e")

```
1 import java.util.Scanner;
2
3 public class Q4 {
4
5
6
7     public static void main(String[] args) {
8
9         while(true){
10
11             Scanner in = new Scanner(System.in);
12             System.out.println("Enter a String:");
13             String input = in.nextLine();
14             System.out.println("Enter a character to be remove:");
15             String character = in.nextLine();
16
17             input = input.replaceAll(character, "");
18             System.out.println("Output: " + input);
19
20         }
21     }
22 }
23
```

Output:

localhost:3000 says  
Are you sure you want to save and submit?  
(Session will end)

Figure 4-24. Attempt submit all solution

At the end of all the assessment questions, there is a button to submit all the answers and obtain the score for each question. The section below describes the process in which the score is computed for the solutions and how the report is generated for the user.

## 4.2.5 Attempt Assessment Questions – Compute Score and Generate Report

### Overview

To perform an assessment of the student's submitted program, the system will first run the submitted programs and then simulate user input to these programs and check if the output corresponds to the expected output. For each question, the simulated user input and expected output are stored in a question specific configuration file.

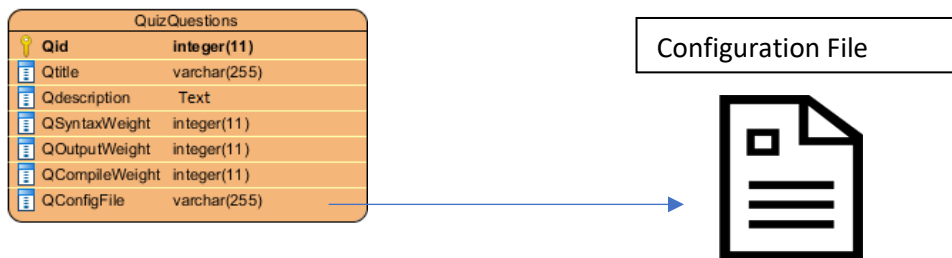


Figure 4-25 Relationship between a QuizQuestion and its corresponding configuration file

These configuration files contain the Title, Inputs, Outputs, Keywords, and Score for each corresponding keyword. Each of the fields is separated using ',' comma as the delimiter.

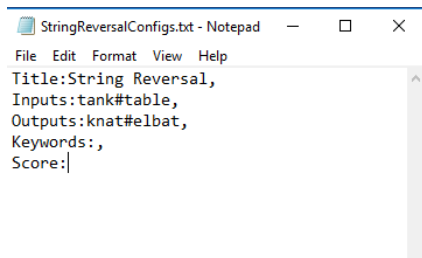


Figure 4-26: Example. Content of a question configuration file

**Title:** Question Title of the corresponding program

**Inputs:** Simulated inputs to pass into student's submitted program. "#" hex separates multiple inputs

**Outputs:** Expected output from the student's submitted program. "#" hex separates multiple outputs

**Keywords:** Keywords that are expected to be found in the student submitted program

**Score:** Marks given for specific keyword found in student's code

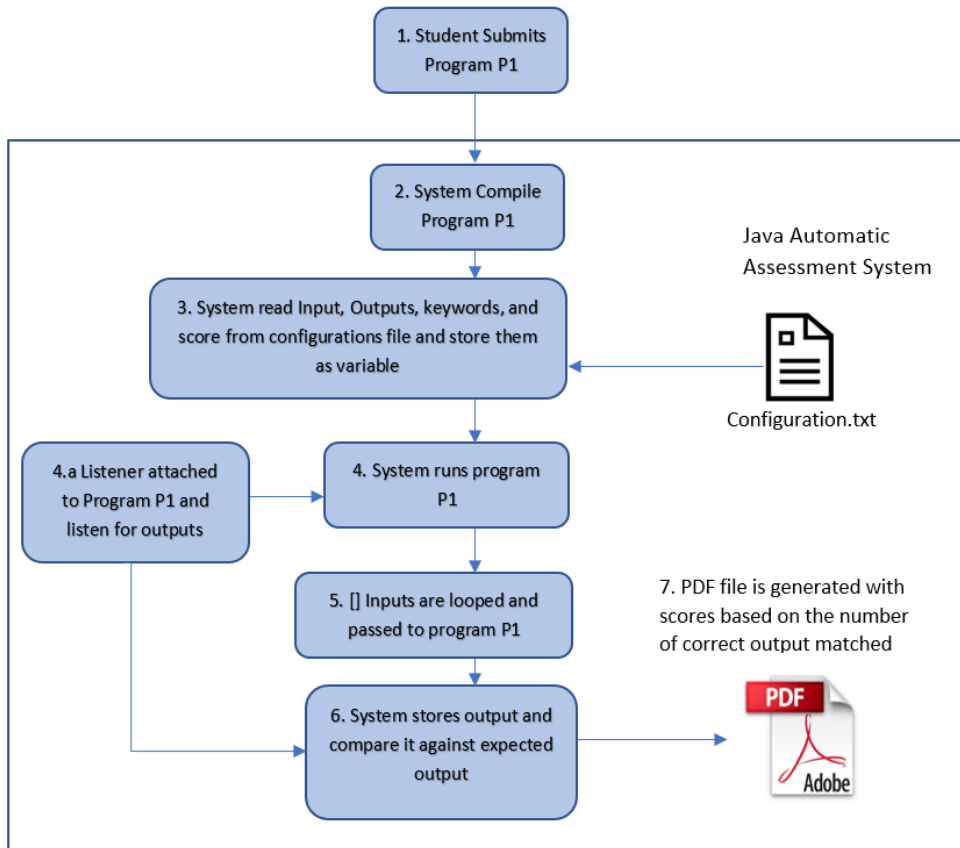


Figure 4-27 Process of assessing student's program

The 'child\_process' package will be used to spawn an instance of the java program and assigned a variable as a reference to it. (Step 4 of figure 4-27)

```
var qlchild = spawn('java', ['-classpath', 'Quiz/Student', 'Q1']);
```

A listener will then be attached to this program and check for its output.

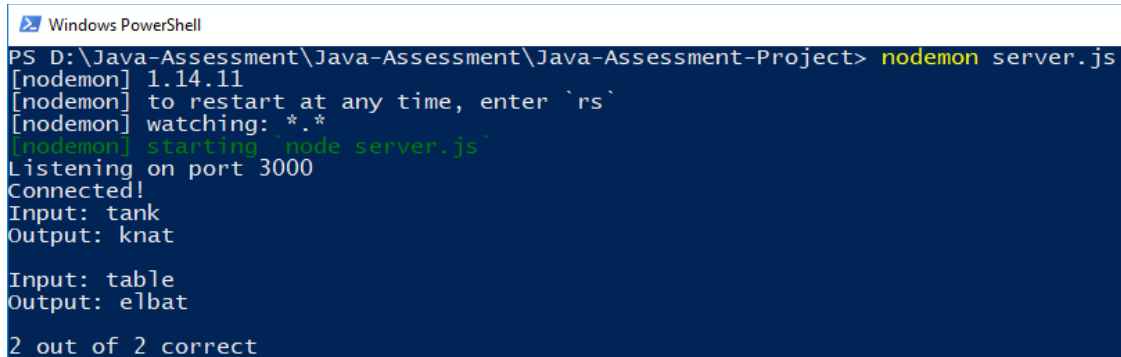
```
qlchild.stdout.on('data', function (data) {  
  
    if(data.toString().includes("Output: ")) {  
        console.log(data.toString());  
        resultseq.push(data.toString());  
        qlchild.stdout.clear();  
    }  
  
});
```

Figure 4-28. Listening to stdout(output) of a program

In Step 5 of Figure 4-27, simulated input that were previously read from the question configuration file are passed to the program

```
for(var j=0;j<inputs.length;j++){  
  console.log(inputs[j]);  
  qlchild.stdin.write(inputs[j]+"\\n");  
}
```

Figure 4-29. Array of input is looped, and each input is sent to the program.( tank, table)



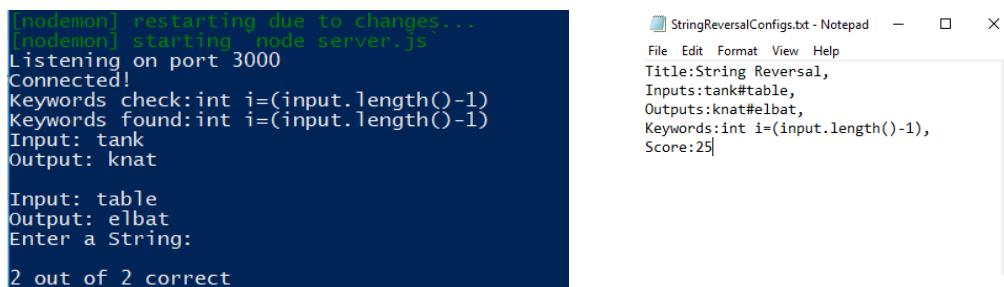
```
Windows PowerShell  
PS D:\Java-Assessment\Java-Assessment\Java-Assessment-Project> nodemon server.js  
[nodemon] 1.14.11  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching: *.*  
[nodemon] starting `node server.js`  
Listening on port 3000  
Connected!  
Input: tank  
Output: knat  
  
Input: table  
Output: elbat  
  
2 out of 2 correct
```

Figure 4-30. Resulting input(tank, table) and output(knat,elbat) are shown in the terminal( e.g. String reversal question)

In addition to checking the output, the system also checks for specific keywords in the student's code, to prevent students from hard-coding their output.

```
for(var e=0;e<keywords.length;e++){  
{  
  if(qlcode.includes(keyword[e])){  
    keywordmatch++;  
    keywordthatmatch.push(keyword[e]);  
    keywordScore += score[e];  
  }  
}
```

Figure 4-31. Loop through a list of keywords and check if they exist in student's code



```
[nodemon] restarting due to changes...  
[nodemon] starting `node server.js`  
Listening on port 3000  
Connected!  
Keywords check:int i=(input.length()-1)  
Keywords found:int i=(input.length()-1)  
Input: tank  
Output: knat  
  
Input: table  
Output: elbat  
Enter a String:  
  
2 out of 2 correct
```

```
StringReversalConfigs.txt - Notepad  
File Edit Format View Help  
Title:String Reversal,  
Inputs:tank#table,  
Outputs:knat#elbat,  
Keywords:int i=(input.length()-1),  
Score:25
```

Figure 4-32 Keywords checked vs Keywords found displayed in terminal

After checking for the output and keywords, the server will generate a report for the assessment session as shown in Step 7 of Figure 4-27.

Figure 4-33 shows the content of the generated report.

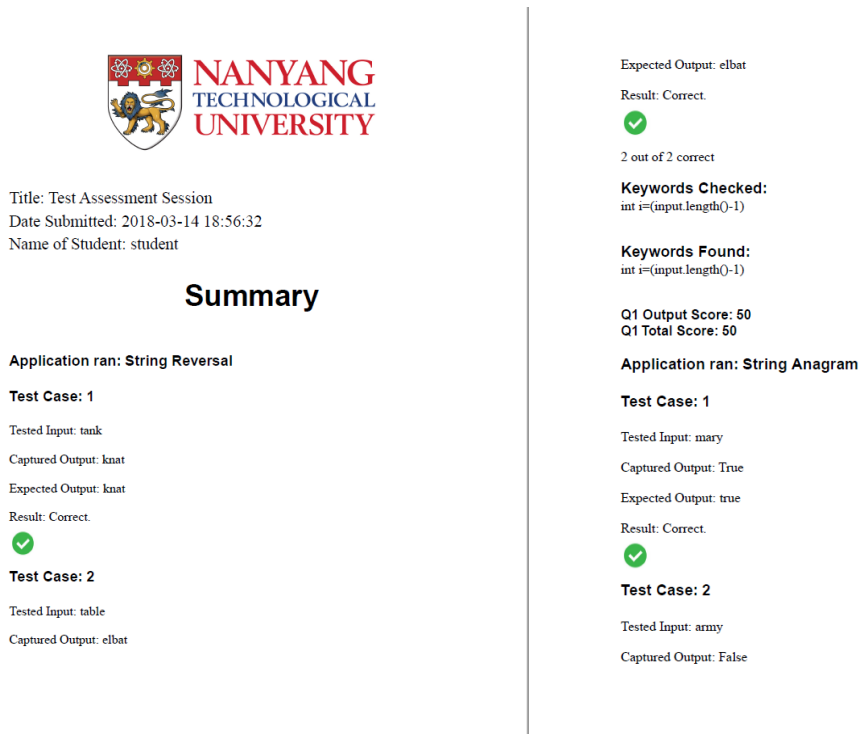


Figure 4-33 Content of the pdf generated by the system

The formula to calculate student's score is as followed:

Total Score of Quiz Session:  $\sum$  Quiz Questions

Quiz Question = Output\_Score + Compile\_Score

Output\_Score = (No. Of Correct Output/ Total No. Of Outputs)\*Output\_Weightage

Compile\_Score(If successful compilation) = (No. Of Correct Keyword/Total No. Of Keywords)\*Compile\_Weightage

Compile\_Score(If unsuccessful compilation) = (No. Of Correct Keyword/Total No. Of Keywords)\*((Syntax\_Weightage/100)\*(Compile\_weightage))

Upon successful submission, the student will be redirected to home page.

## 4.2.6 View Labs

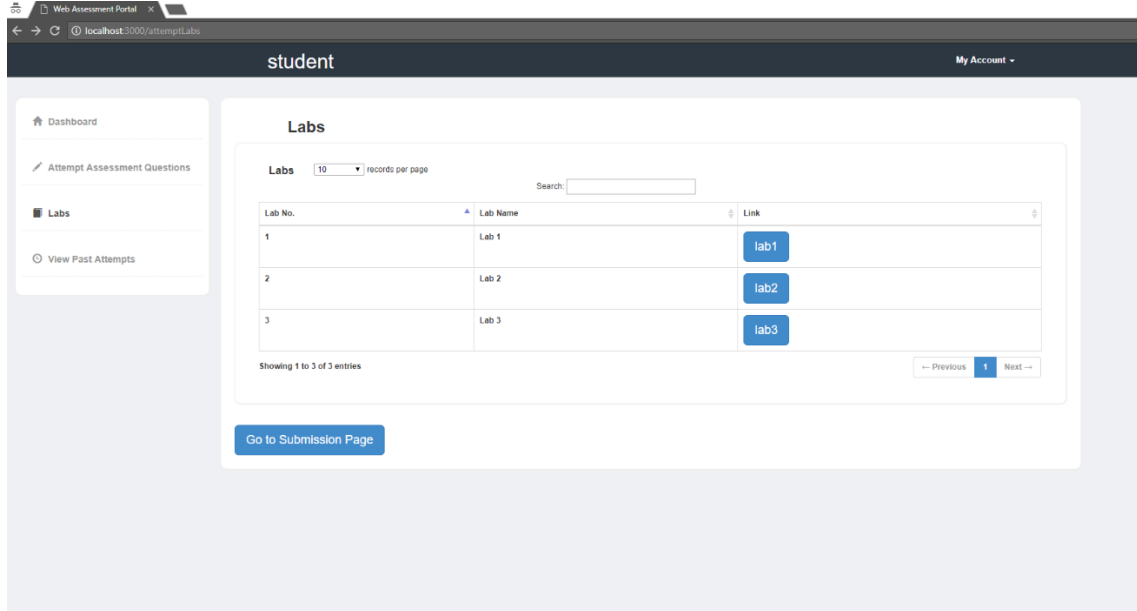


Figure 4-34. View Lab Page. Display a list of all labs from the database

In the Labs tab, students can view existing labs available to them. Upon clicking the Lab buttons, students will be able to view the lab pdf documents.

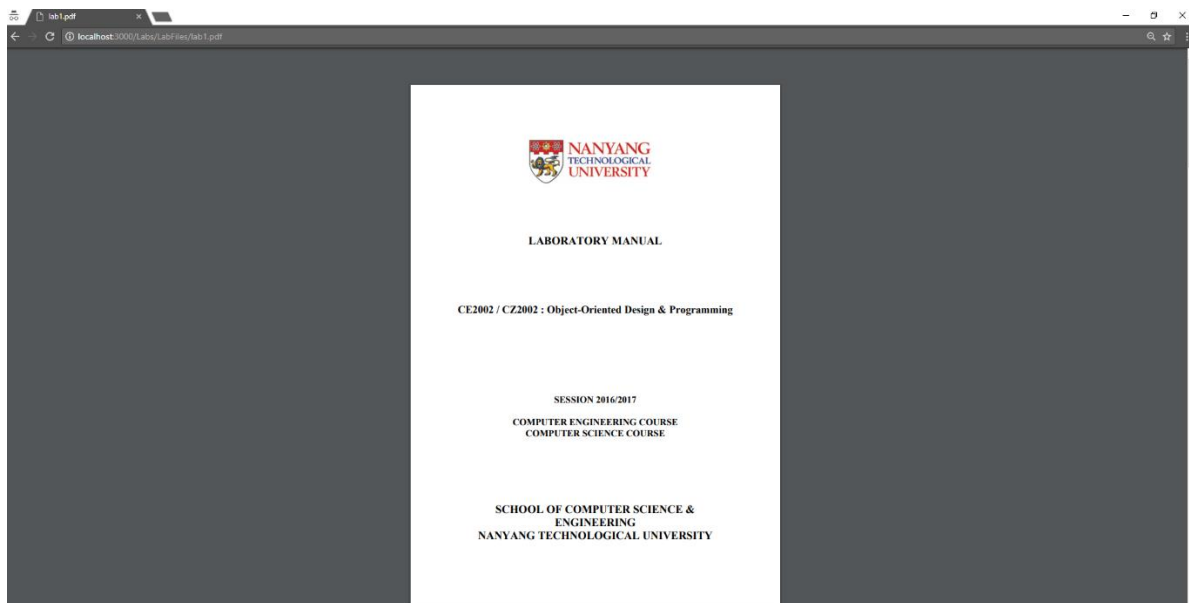


Figure 4-35. Student viewing lab document from the page

Students can also click on the “Go to Submission Page” in Figure 4-44 to go to the lab submission page, (Figure 4-36) where students can submit their lab solution online.



SCE17-0389  
Java Program Auto Assessment

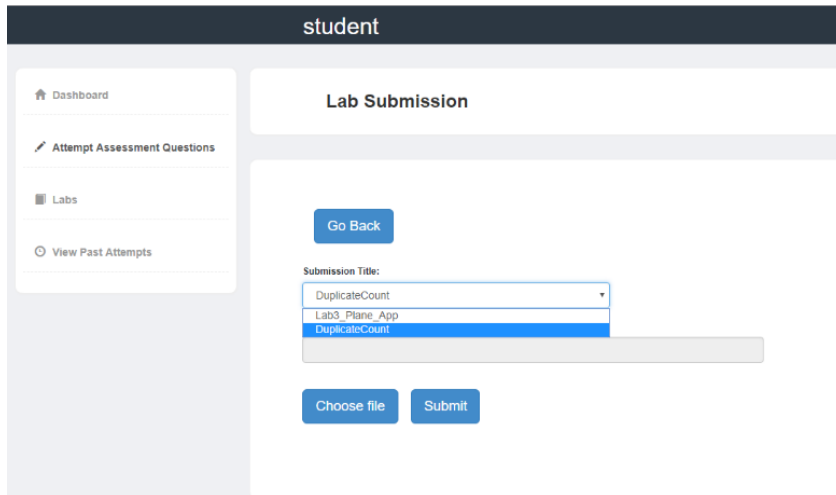


Figure 4-36. Submission of Lab Assignments

In the lab submission page, students will be able to choose from a dropdown list the lab assignment which they wish to submit. They will be given the option to upload their solution file in either .java or .zip format. If the assignment solution consists of multiple files, the student must zip up their project containing their java files.

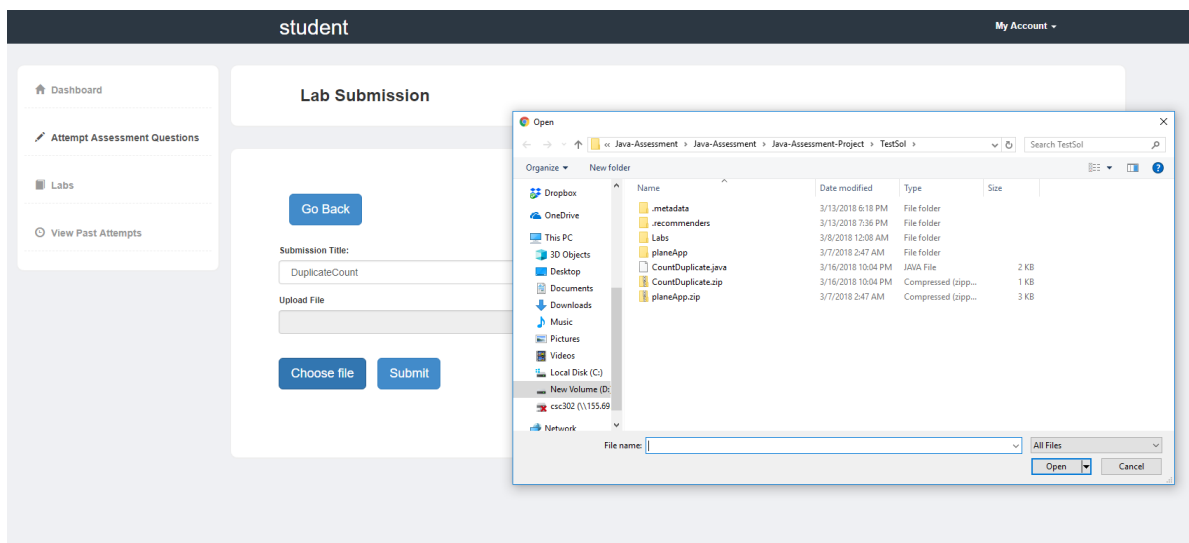


Figure 4-37. Uploading a file for submission

Upon successful upload, the files will be compiled and processed as shown in Figure 4-37. If there are any errors in compiling the java files, the cause of the failure will be printed in the report, so that student will be able to know where the problem is. (Shown in figure 4-38)

## SCE17-0389

### Java Program Auto Assessment

```
Keywords Checked:
charAt

Keywords Found:
charAt

Error(s):
Error: Command failed: javac Labs/student/*.javaLabs/student

\CountDuplicate.java:12: error: ';' expected
whilee(true){
Labs/student/CountDuplicate.java:33: error: 'class' expected
for(int count = 0;count<input.length();count++){
Labs/student/CountDuplicate.java:33: error: > expected
for(int count = 0;count<input.length();count++){
Labs/student/CountDuplicate.java:33: error: not a expected
for(int count = 0;count<input.length();count++){
Labs/student/CountDuplicate.java:33: error: illegal start of expected
for(int count = 0;count<input.length();count++){
Labs/student/CountDuplicate.java:33: error: ';' expected
for(int count = 0;count<input.length();count++){
Labs/student/CountDuplicate.java:49: error: ';' expected
output += " - " + input.charAt(j) + " " + (duplicate);
Labs/student/CountDuplicate.java:49: error: not a expected
output += " - " + input.charAt(j) + " " + (duplicate);
8 errors

-----End of Report-----
```

Figure 4-38. Error message printed in the report

Once the report has been generated, the student will be able to view their past attempts and reports of quizzes and labs in the “View Past Attempts” tab (Figure 4-39).

## 4.2.7 View Past Attempts

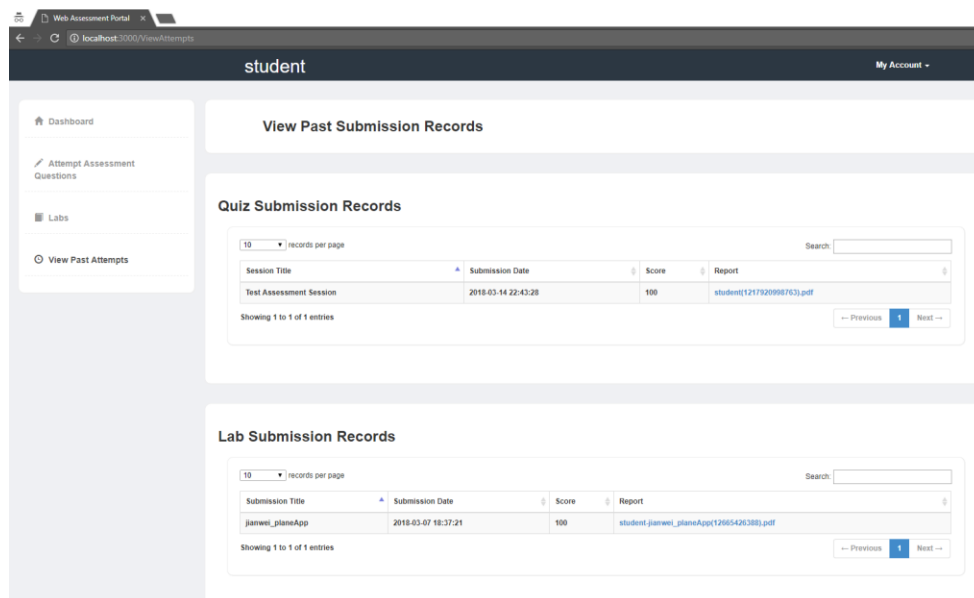


Figure 4-39. Past attempts of student's lab and quiz submissions

Submission Title, Submission Date, Score, and report file location of past attempts are displayed in this page. This information is stored in the QuizSubmissionRecords as well as LabSubmissionRecords tables respectively.

The reports generated are stored in a folder on the server side. Students are also able to click on report's link to view the summary of their attempts and download as pdf to view offline.

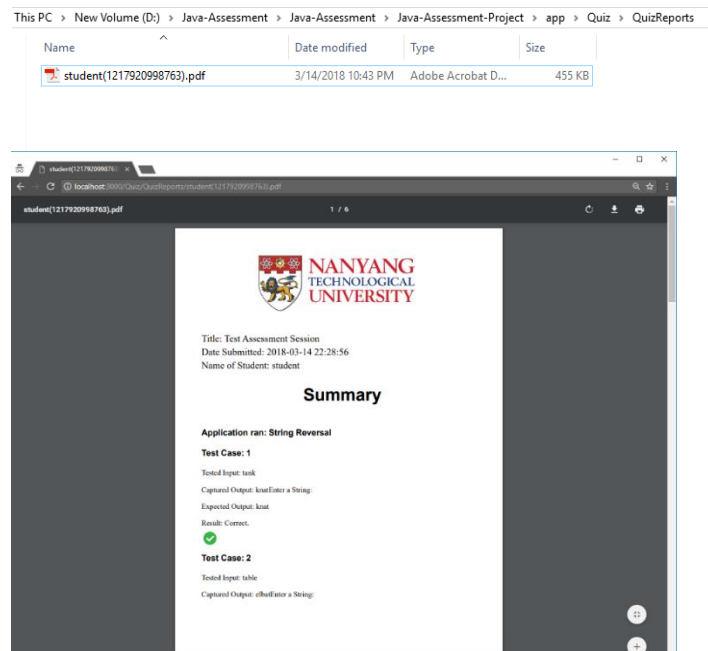


Figure 4-40. Viewing past report online

### 4.3 Admin Functions Implementation

As the following functions have similar implementation and user interface, one example will be chosen to be used as an example in this section.

Create Quiz Session	Delete Quiz Session	Modify Quiz Session
Create Labs	Delete Labs	Modify Labs
Create Annoucements	Delete Annoucements	Modify Annoucements
	Delete Students	Modify Students

## 4.3.1 Create Quiz Session

The 'Create Quiz Session' form includes the following fields:

- Quiz title:
- Description:
- Session ID:
- Session Password:
- Time Limit:
- Create button

Below the form is a 'Session List' table with 10 records per page. The table has columns: Session ID, Session Title, Description, Session Pass, and Time Limit.

Session ID	Session Title	Description	Session Pass	Time Limit
Lab04	Test Assessment Session	This is a test session. This session consist of 4 questions weighing 25 marks each. Please try to attempt all questions. Click the button below to start your attempt. Maximum of 1 attempt(s).	123456	90

Figure 4-41. Create Session user interface

Every Create function (e.g Create Lab, Create Student, Create Question etc) will have a UI that allows the admin to input the required fields and the information will be POST to the server-side and written to the database. At the bottom of the page, there will be a table displaying a list of existing sessions.

```
exports.createSessionInDB = function(req,res){
  db.query('INSERT INTO sessions (Sid, Stitle, Sdescription, Spass, TLimit) VALUES (?, ?, ?, ?, ?)', [req.body.data.Sid, req.body.data.sessionTitle, req.body.data.quizDescription, req.body.data.sessionPass, req.body.data.timeLimit], function (error, results, fields) {
    if(error){
      console.log('Code 400, Error occurred');
      console.log(error);
      res.json({ state: 0 });
    }
    else{
      console.log("success");
      res.json({ state: 1 });
    }
  });
}
```

Figure 4-42. Code: server-side function to write session details into database

## 4.3.2 Delete Quiz Session

The 'Delete Session' form includes the following fields:

- Enter Session ID:
- Delete button

Below the form is a 'Session List' table with 10 records per page. The table has columns: Session ID, Session Title, Description, Session Pass, and Time Limit.

Session ID	Session Title	Description	Session Pass	Time Limit
Lab04	Test Assessment Session	This is a test session. This session consist of 4 questions weighing 25 marks each. Please try to attempt all questions. Click the button below to start your attempt. Maximum of 1 attempt(s).	123456	90
Quiz01	Test Assessment Session	A Test Session	123456	60

Showing 1 to 2 of 2 entries

Figure 4-43. Delete Quiz User Interface

To delete an existing quiz session, admin is required to enter the session ID in the field. If the session ID exists, a query will be executed from the server to delete the specific session.

### 4.3.3 Modify Quiz Session

The screenshot displays the 'admin' interface for modifying a quiz session. It includes a sidebar with navigation links, a 'Search Session' form, a 'Modify Session' form, and a 'Session List' table.

**Search Session**

Enter Session ID:

**Modify Session**

Session ID:

Session Pass:

Session Title:

Session Description:

Session Time Limit:

**Session List**

Session ID	Session Title	Description	Session Pass	Time Limit
Lab04	Test Assessment Session	This is a test session. This session consist of 4 questions weighing 25 marks each. Please try to attempt all questions. Click the button below to start your attempt. Maximum of 1 attempt(s).	123456	90

Figure 4-44. Modify Session User Interface

To modify existing session's details, e.g. Session ID, pass etc., the admin must enter the session ID and press search. The information will then be pulled out from the database and displayed at the page to be modified.

### 4.3.4 Create Assessment Questions

The screenshot displays the 'admin' interface for creating a quiz assessment question. It includes a sidebar with navigation links, an 'Add Assessment Question' form, and a 'Test Config File' form.

**Add Assessment Question**

☒ Quiz Question ☐ Lab Question

Question Name:

Description:

Syntax Weightage:

Output Weightage:

Compile Weightage:

☒ Upload Config File ☐ Manually Specify

Upload config file:

**Test Config File**

Upload a Solution File to Test:

File: CountDuplicate.zip  
100%

Figure 4-45. Creating a Quiz Assessment Question

Admin can toggle between creating a lab or a quiz question using the radio button. To create a quiz question, the admin must specify the question title, description, syntax weightage, output weightage as well as the compile weightage (Syntax weightage is a subset of compile weightage). Admin must also either upload the configuration file for the question or manually create one using the interface, as shown in Figure 4-46.

The screenshot displays the 'Add Assessment Question' interface. On the left is a sidebar with navigation links: Dashboard, Manage Announcements, Manage Sessions, Manage Labs, View Students Result, Manage Assessment Questions, and Manage Student Records. The main content area has two tabs: 'Quiz Question' and 'Lab Question'. The 'Lab Question' tab is active, showing a form for adding a question. The 'Question Name' field contains 'DuplicateCount'. Below it are three weightage sliders: 'Syntax Weightage' (50), 'Output Weightage' (50), and 'Compile Weightage' (50). There are two radio buttons: 'Upload Config File' (selected) and 'Manually Specify'. Under 'Upload Config File', there is a 'File name' input field and four text areas for 'Input', 'Output', 'Keywords', and 'Score', each with a note about using 'F' as a delimiter. At the bottom are 'Create Config File' and 'Create' buttons. To the right is a 'Test Config File' panel with a 'Choose file' button and a 'Test' button.

Figure 4-46. Creating a Lab assessment question - creating a config file using the interface

To manually create the configuration file, the admin must enter the configuration file name, specifying the simulated inputs, expected output and keywords for the configuration file. Scores can also be specified for each keyword.

By clicking on the “Create Config File button”, the configuration text file will be created using the specified inputs.

Before the admin can create the question, it is highly recommended for the admin to test the configuration file which was previously uploaded or created. This can be done using the “Test Config File” button which allows the admin to upload a solution program and simulate the run

SCE17-0389  
Java Program Auto Assessment

using the configuration's input and output. At the button page of the interface, there is also a log screen where admin can see if there are any issues when running the configuration test. If there is no error, a "Test Successful" message will be displayed in the log.

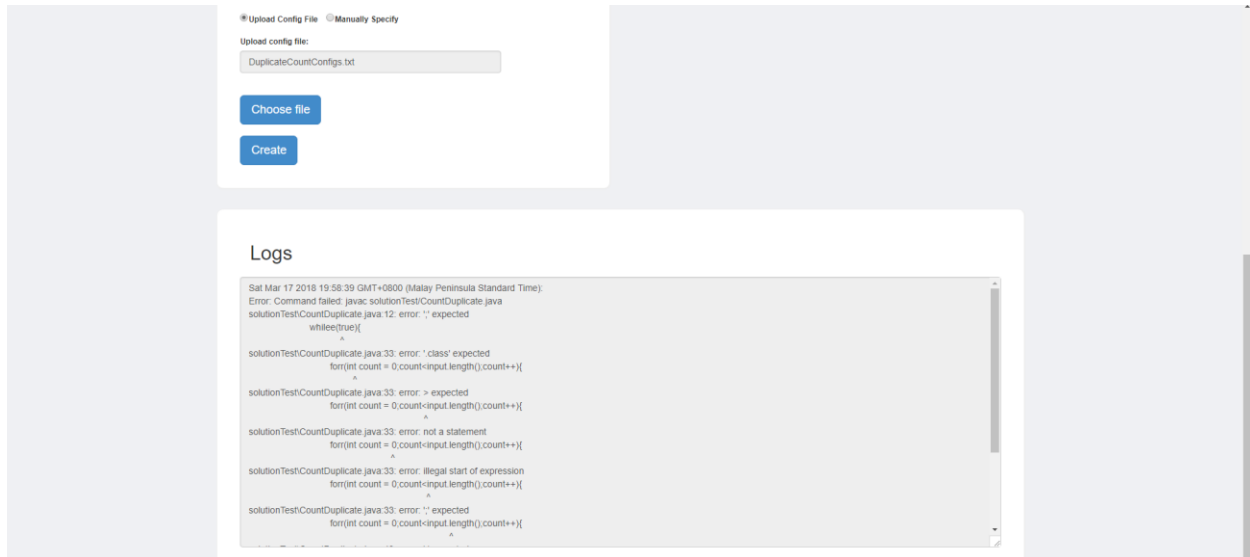


Figure 4-47. Error message showing the solution program contains error and the configuration file was not able to run

### 4.3.5 Add students

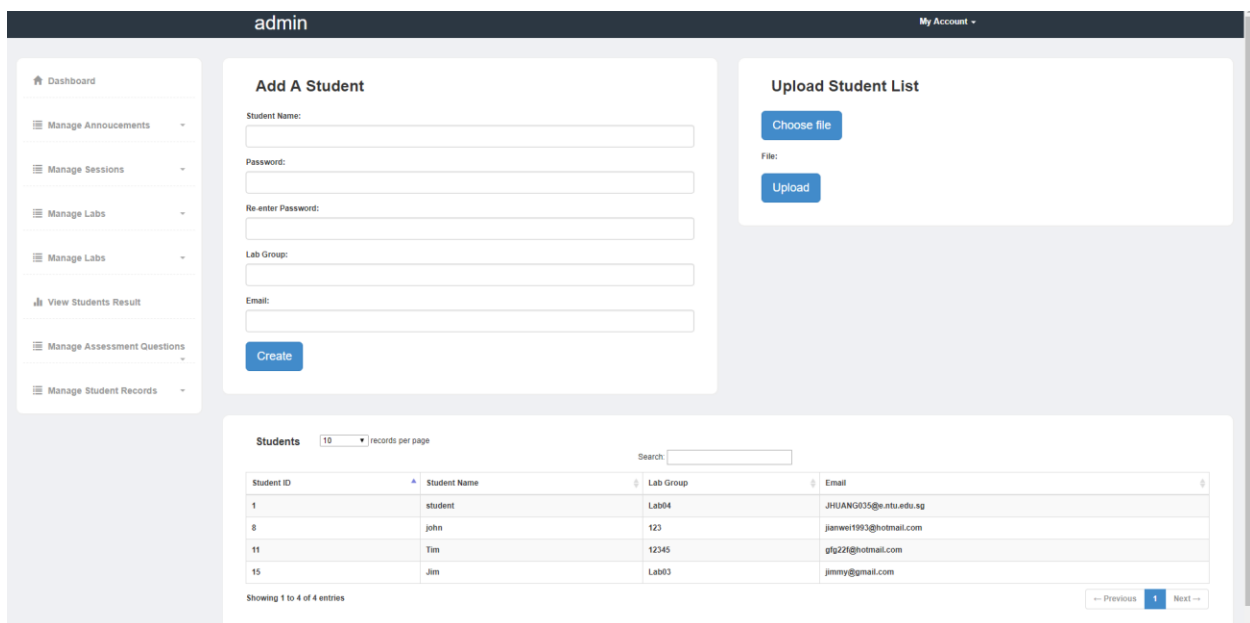
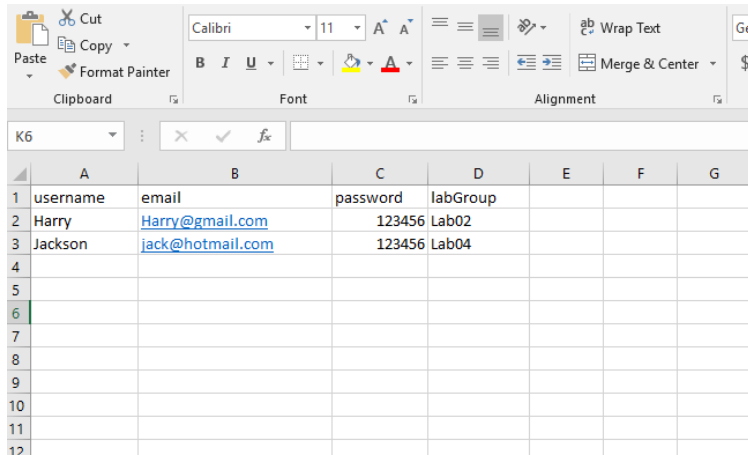


Figure 4-48. User Interface for adding student

Admin can choose to create a single student account or upload a list of students from an excel sheet.

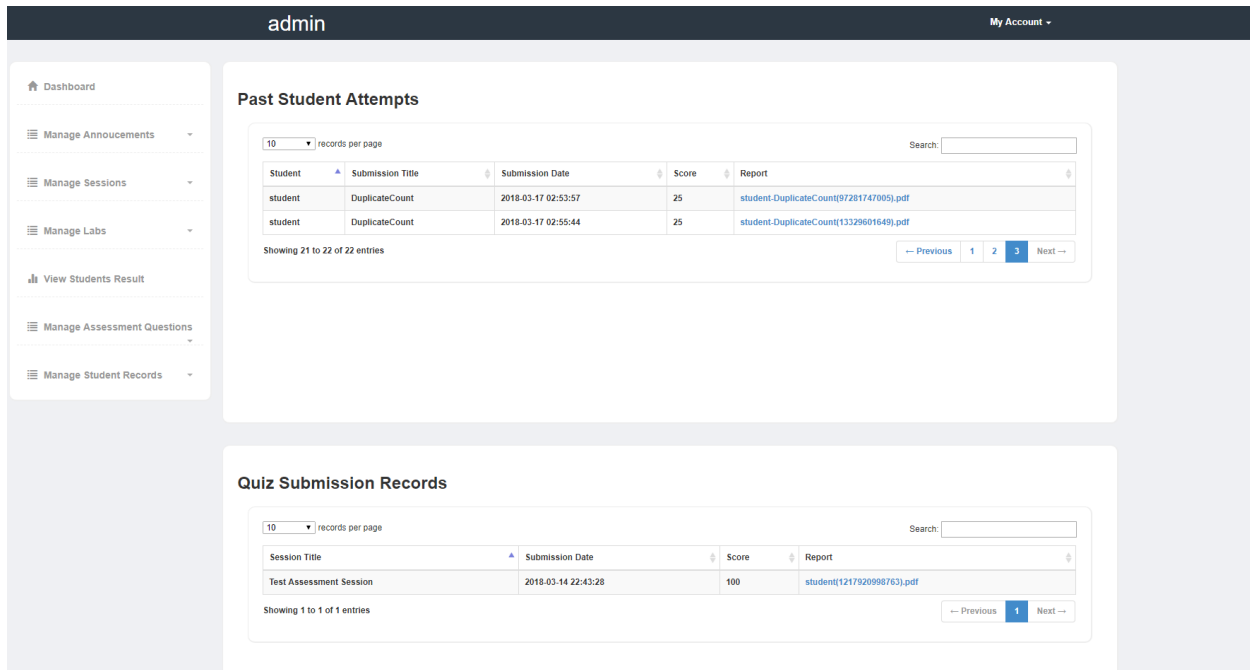
The excel sheet details must follow the order as shown in Figure 4-49.



	A	B	C	D	E	F	G
1	username	email	password	labGroup			
2	Harry	Harry@gmail.com	123456	Lab02			
3	Jackson	jack@hotmail.com	123456	Lab04			
4							
5							
6							
7							
8							
9							
10							
11							
12							

Figure 4-49. Excel format if admin choose to upload a list of students

### 4.3.6 View Student Attempts



The screenshot shows the admin dashboard with a sidebar on the left containing navigation links: Dashboard, Manage Announcements, Manage Sessions, Manage Labs, View Students Result, Manage Assessment Questions, and Manage Student Records. The main content area has two sections:

#### Past Student Attempts

10 records per page

Student	Submission Title	Submission Date	Score	Report
student	DuplicateCount	2018-03-17 02:53:57	25	<a href="#">student-DuplicateCount(97281747005).pdf</a>
student	DuplicateCount	2018-03-17 02:55:44	25	<a href="#">student-DuplicateCount(13329601649).pdf</a>

Showing 21 to 22 of 22 entries

Navigation: Previous, 1, 2, 3, Next

#### Quiz Submission Records

10 records per page

Session Title	Submission Date	Score	Report
Test Assessment Session	2018-03-14 22:43:28	100	<a href="#">student(1217920998763).pdf</a>

Showing 1 to 1 of 1 entries

Navigation: Previous, 1, Next

Figure 4-50. Viewing all past attempts on the admin page

Lastly, admin has a page where they can view past attempts of all students for both their Quiz submissions as well as lab submissions. The data are displayed using the information in both quizsubmissionrecords as well as labsubmissionrecords table.



## 5. Limitations and Difficulties Faced

Throughout the development of the web application, there were many setbacks in implementing some of the functions that was initially thought of, and the author had to come up with alternative solutions.

This section elaborates the problems and issues faced during the development of the Java Automatic Assessment System.

### **Difficulty in choosing the Web Application Stack, Language used**

As there is no silver bullet in the field of software engineering, there was much uncertainty in the initial stage of the project as to which framework and language were the most suitable for this project. Nodejs, AngularJS, and MySQL were all chosen due to the number of libraries and support readily available on the web. As this is the first web-application built by the author using the combination of Nodejs and Angular, the initial learning curve was quite steep.

### **Inexperience in designing User Interface/User Experience**

Throughout the development of this system, more emphasis was given to making functions work instead of designing a more intuitive and resilient user interface. This is due to fact that the UI was developed with the point of view of a developer instead of an end user who has no knowledge or background programming and how the system work. With the help of his supervisor, the UI was improved over many iterations through the course of the project.

### **Powerful Libraries, but incomprehensive examples**

Majority of the libraries and package provided comprehensive documentation. However, there were some that were lacking in examples and guide on usage. As a result, there were many attempts at trial and errors trying to use these packages.

### **Working with Asynchronous calls**

Although there were many advantages in using an asynchronous framework like NodeJS, there were also some areas where there was race condition occurs when asynchronous calls were used. In addition, there were some libraries that do not support synchronous calls. One workaround was to implement a sleep function at a specific checkpoint to ensure that functions are sync,

## **6. Conclusion and Future Recommendations**

### **6.1 Conclusion**

The primary objective of this Final Year Project is to create a web-based application with a server support that can assess student's programs and codes. The author believes that the Java Automatic Assessment System have achieved that goal, through comparing programs' output and checking for keywords found in codes against a set specified by the admin.

Even though the system has been developed, it has not yet gone through live runs and is still considered a new, akin to a working prototype. There is still more testing to be performed to identify areas where improvement can take place.

At the rate of technology advancement, it is also worth noting that there are also possibilities that the system could be improved by either replacing the framework, language or the libraries and package used.

A full stack web application such as the Java Automatic Assessment System may be difficult to implement due to the use of multiple languages and frameworks, but it was truly an enriching journey, offering an opportunity to apply what was taught during the course of 3 years.

## 6.2 Future Recommendation

The author has identified areas where the system could be improved and could be used as reference for future developers.

### **Exploring different languages and framework**

The current web-application stack consists of AngularJS, NodeJs, and MySQL. One issue that was noted while using AngularJS is that during data-binding, data were not yet retrieved from the server. Thus, some alternative to using AngularJS are ReactJS, Ember, and backbone.js.

Although asynchronous framework has its advantages, there some operations that were just not suitable. Many a times there were race conditions existing at the backend, causing the author to set delay to certain functions. Alternative language like PHP or Java may not face this issue. In addition, different languages provide different libraries and packages, hence the performance of system built on different libraries may improve.

### **Enhancement of User Interface**

As functionalities were prioritized over aesthetics during the implementation, there are still aspects of the user interface that requires optimization. For example, fewer steps may be required to achieve a task.

### **Expanding Project Scope**

The current system can only assess Java programs and codes, however, this could be possibly be extended to other programming languages like C, C++ or python. To support the assessment of other language, the user interface will have to be revamped as well.

### **Adding Additional Features**

The two features below are ideas that could provide an improved quality of life for the users but was not able to be implemented due to the time constraint for the duration of this project.

### **Edit Account Function**

Every user should be able to change their account password at any point of time. The current user interface has a profile button which can be used bring users to a page where they can change their account information.

### **Automatic Email System**

There were also ideas to create an automatic email function where the system will email the generated report and result to the student. The system can also leverage the email system to notify student that there is a new assignment or notify admin that a specific student has attempted a submission.

## Appendix A - Directory of Project Files

The root directory of the JAAS web application contains the following folder and files:

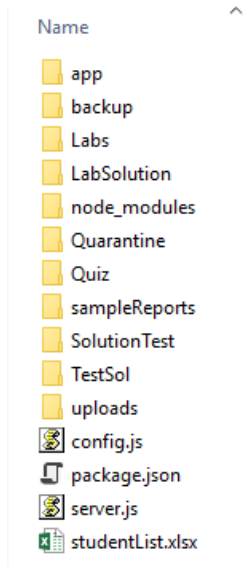


Figure A-1. Files and Folders in the root directory

**app Folder** – Contains the model, view, and controller files of the web application. The app folder also contains a sub-folder, called the “Labs” folder which contains each lab’s document as well as a folder which stores all generated student’s report for lab submissions. The quiz sub-folder contains code submitted by the students for their quiz session as well as the report generated for their quiz submission.

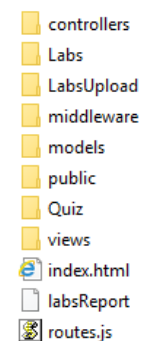


Figure A-2. Sub-folders in app folder

## Views Directory

The view files are organized into admin and student views. All views are programmed using HTML.

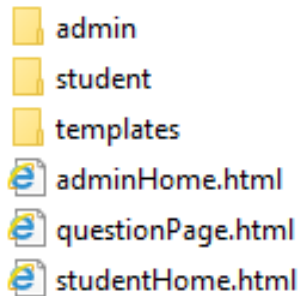


Figure A-3 View folders

### Admin Views

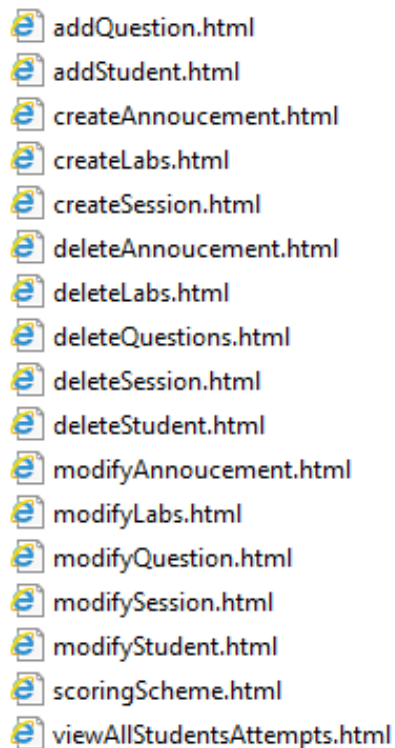


Figure A-4 Admin pages

### Student Views

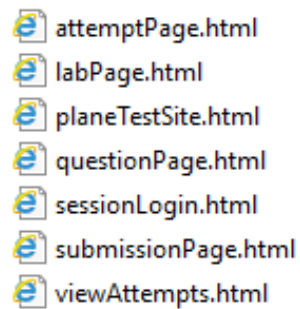


Figure A-5. Student Pages

## Controller Directory

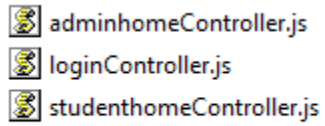


Figure A-6 Controllers files for JAAS

The controller files provide routing to the functions at the server side.

## Model Directory

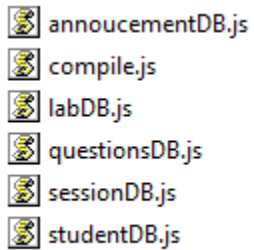


Figure A-7 Model files for JAAS

The model files contain functions that interacts directly with the database. For example, the studentDB.js contains functions like addStudents(), deleteStudents and modifyStudents etc.

**configs.js** – contains the configuration file for Express.js to connect the database. Host, user, password, and database are defined in this file.

(Refer to Figure 3-4.)

**package.json** – defines the list of packages that are used by the project.

---

```
{
  "name": "jianwei",
  "version": "1.0.0",
  "description": "This is a Java Auto Assessment system",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "JianWei",
  "license": "ISC",
  "dependencies": {
    "angular-file-upload": "^2.5.0",
    "angular-module-core": "^1.2.19",
    "angular-resource": "^1.6.6",
    "angular-route": "^1.6.6",
    "async-function-queue": "^1.0.0",
    "bcrypt-nodejs": "0.0.3",
    "body-parser": "^1.17.2",
    "case-insensitive": "^1.0.0",
    "childprocess-queue": "^1.0.1",
    "delay": "^2.0.0",
    "exceljs": "^0.6.0",
    "express": "^4.15.4",
    "express-session": "^1.15.5",
    "extract-zip": "^1.6.6",
    "jsonwebtoken": "^7.4.3",
    "mongoose": "^4.11.8",
    "morgan": "^1.8.2",
    "multer": "^1.3.0",
    "mysql": "^2.14.1",
    "ng-file-upload": "^12.2.13",
    "node-async-loop": "^1.2.2",
    "nodemon": "^1.14.11",
    "passport": "^0.4.0",
    "pdfkit": "^0.8.3",
    "router": "^1.3.1",
    "sequelize": "^4.8.0"
  }
}
```

Figure A-8. Contents of package.json, list of packages used for this project



**server.js** – contains the script to run express as a web server. The Port used by the web server also can be changed in this file.

```
var express = require('express');
var bodyParser = require('body-parser');
var morgan = require('morgan');
var db = require('./config');
var multer = require('multer');
var session = require('express-session');
var upload = multer({ dest: 'uploads/' });

var app = express();

app.use(bodyParser.urlencoded({extended:true}));
app.use(bodyParser.json());
app.use(morgan('dev'));
require('./app/routes')(app);

app.use(express.static(__dirname + '/app'));

app.listen(3000,function(err){
  if(err){
    console.log(err);
  }
  else{
    console.log("Listening on port 3000");
  }
});
```

Figure A-9. Configurations for the server

**routes.js** – Express.js uses routing to handle GET/POST request from the client. The application uses routes to listen to the corresponding URL request. For example, if client requests for /createAnnouncement URL, it will be redirected to the correct callback function via the routing table. All routes are defined in this file.

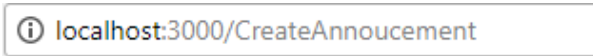


Figure A-10. POST URL

```
app.post('/CreateAnnouncement',function(req,res){  
    res.sendFile(__dirname + '/views/admin/createAnnouncement.html');  
  
});
```

Figure A-11. A POST request returning a resource file

## To start the Express server

Navigate to the root folder -> shift+ right click and choose “open command prompt here”. To start, simply enter “nodemon server.js”.

```
PS D:\Java-Assessment\Java-Assessment\Java-Assessment-Project> nodemon server.js  
[nodemon] 1.14.11  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching: *.*  
[nodemon] starting 'node server.js'  
Listening on port 3000  
Connected!
```

Figure A-12. Starting the web application using nodemon/cmd

## Appendix B - Functional Testing

The objective of performing functional testing is to ensure that every function is working as intended. Black-box testing was applied during the testing process to ensure that testing is balanced and unprejudiced [10]. Another advantage is that black-box testing can help to identify vagueness, especially in UI implementation, as the UI must be as intuitive as possible.

The table below shows the functions that were tested during the testing stage.

### Student

Objective	Description	Expected Result	Test Result
Login	User enter credentials	User enters home page	Pass
View Annoucements	User enter home page	Annoucements displayed in home page	Pass
View Labs	User click on “View Labs” tab	List of labs displayed	Pass
Submit Labs	User upload a .java file or a .zip file containing the .java files	A report will be generated after a successful submission and submission attempt will be recorded	Pass
Log in Assessment Session	User enters session ID pass	User directed to session question page	Pass
Attempt Assessment Questions	User enters solution for questions and submit	A report will be generated after a successful submission and submission attempt will be recorded	Pass
View Past Attempts	User view past attempts on “View Past Attempts” page	User able to see .pdf reports of past attempts	Pass

## Admin

Objective	Description	Expected Result	Test Result
Login	User enter credentials	User enters admin home page	Pass
Create Announcements	User enter title, description, and lab group of the announcement	Announcements created in the database	Pass
Delete Announcements	User enter announcement ID to be delete	Announcements is deleted from database	Pass
Modify Announcements	User update announcements	Announcement updates is reflected in database	Pass
Create Lab	User enter lab No., lab Title and upload lab document	Lab created and reflected in database	Pass
Delete Lab	User enters lab No.	Lab will be deleted from database	Pass
Modify Lab	User update lab details	Lab updates is reflected in database	Pass
Create Session	User enter session title, description, session ID, session pass and time limit	Session is created in the database	Pass
Delete Session	User enter Session ID to be delete	Session is deleted from database	Pass
Modify Session	User update session details	Session updates is reflected in database	Pass
Create Student	User enter student details or upload a list of students from excel	Student records created in database	Pass
Delete Student	User enter student email to be delete	Student is deleted from database	Pass

SCE17-0389  
Java Program Auto Assessment

Modify Student	User enters student details	Student is updated in the database	Pass
Create Assessment Questions	User enter required details to create assessment question	Assessment Question and config file is created	Pass
Delete Assessment Questions	User enter question ID to be removed	Assessment Question is deleted from database	Pass
Modify Assessment Questions	User modify a assessment question's details	Changes are reflected in database	Pass
View Students Attempts	User click on "View all Students Attempt" tab	User able to see list of all student lab's submission attempts and quiz submission attempts	Pass

## Appendix C - References

- [1] N. Boudewijn, "Automated Grading of Java Assignments," Master of Science Master, Faculty of Science, Graduate School of Natural Science, UTRECHT UNIVERSITY, 14 July 2016.
- [2] Adam Khalid, "Automatic Assessment of Java Code". The Maldives National Journal of Research, The Maldives National University. Vol 1, No 1, pp 732 June 2013
- [3] G. Sim, P. Holifield, and M. Brown, "Implementation of computer assisted assessment: lessons from the literature," *Alt-J*, vol. 12, no. 3, pp. 215-229, 2016.
- [4] N. Yusof, N. A. M. Zin, and N. S. Adnan, "Java Programming Assessment Tool for Assignment Module in Moodle E-learning System," *Procedia - Social and Behavioral Sciences*, vol. 56, pp. 767-773, 2012.
- [5] H. Kitaya and U. Inoue, "An Online Automated Scoring System for Java Programming Assignments," *International Journal of Information and Education Technology*, vol. 6, no. 4, pp. 275-279, 2016.
- [6] Mike Joy, Nathan Griffiths and Russell Boyatt, "The BOSS Online Submission and Assessment System", University of Warwick, United Kingdom. ACM Jeric, Vol. V, No. N, September 2005.
- [7] "A Review of CAL Packages: Ceilidh" np nd Web. Available: [http://www.doc.ic.ac.uk/~nd/surprise\\_95/journal/vol2/hst/article2.html#abstract](http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol2/hst/article2.html#abstract) [Assessed 23-March-2018]
- [8] S. D. Bedford, E. K. Burke, E Foxley and C. A. Higgins, "Ceilidh: A Courseware System for the Assessment and Administration of Computer Programming Courses in Higher Education", Learning Technology Research Computer Science Department University of Nottingham NOTTINGHAM NG7 2RD, UK nd Web. Available: [http://www.cs.joensuu.fi/~mtuki/www\\_clce.270296/Burke.html#System](http://www.cs.joensuu.fi/~mtuki/www_clce.270296/Burke.html#System) [Assessed 23-March-2018]
- [9] Riku Saikkonen, Lauri Malmi and Ari Korhonen, "Fully Automatic Assessment of Programming Exercises", Department of Computer Science and Engineering, Helsinki University of Technology Finland 2001
- [10] Koundinya, "Black Box Testing, Its Advantage and Disadvantages", Web. 25 Jun 2010 Available: <https://www.codeproject.com/Articles/5579/Black-Box-Testing-Its-Advantages-and-Disadvantages> [Assessed 23-March-2018]
- [11] Microsoft. "Chapter 2: Key Principles of Software Architecture" Available: <https://msdn.microsoft.com/en-us/library/ee658124.aspx> [Assessed 23-March-2018]