TESTSQL PROJECT REPORT

Phumezile Tsitsa

Kane Gibson

Alex Ikin

Client - Sonia Berman, sonia@cs.uct.ac.za

Tutor Dennis Hammerschlag, hmmden001@myuct.ac.za

Contents

Abstract

TESTSQL is a system to create and mark SQL assignments. It consists of a database of questions (data access requests in English) and answers (SQL SELECT statements to retrieve that data). The lecture will feed the system with question-pair and the assignment will then generate more based on the examples given. It generates a set of assignments, one for each student in the class, the assignment each student is given is derived from their student number. Each assignment consists of a fixed number of generated questions, spread equally across difficulty levels for each student. The system also stores in its database the data to be queried. Students can use the system to submit their assignment which is then marked automatically against a fixed expected output.

Introduction

TESTSQL is a comprehensive system that empowers second-year computer science lectures with tools necessary to create and automate SQL assignments for their database courses. The system streamlines the process of creating assignments by allowing lecturers to tailor assessments based on desired criteria. With the aid of an efficient database, the system will generate unique assignments for each student, ensuring fairness and combatting cheating. Additionally, the system includes an automatic grading component that evaluates and provides feedback to student submissions. This helps improve the overall efficiency and value of the SQL assignment management process at the UCT (University of Cape Town) (University of Cape Town) Computer Science department.

Software Development Methodology

Our software development process followed a test-driven approach, combining elements from both the waterfall and agile software development methodologies while incorporating SCRUM techniques. The project was divided into several distinct stages/iterations, each building upon the previous one. Every stage had its own predefined deliverables and typically spanned a one-week duration, apart from the implementation and testing stage, which lasted for a month.

At the start of each week, our team convened to assign tasks to individual members. Throughout the week, daily meetings were held at 10 am, providing each team member with the opportunity to showcase their progress up to that point and to report any challenges they encountered while working on their assigned tasks, which could potentially impact the project's success. These meetings served as a platform for sharing ideas on how to address issues and offered valuable advice to enhance the quality of our work.

During the initial stage, one of our key deliverables was the development of a comprehensive test plan. This plan played a pivotal role as it was intended to be utilized for testing the system throughout all subsequent stages. Its primary objectives were to ensure software quality and reduce the occurrence of bugs to an absolute minimum (aiming for 100% bug reduction).

At the conclusion of each iteration, the artifacts generated were shared and discussed with both the client and our tutor. If any issues or discrepancies were identified in the documentation, our team would return to the planning phase to rectify and refine any necessary aspects. This process ensured that our client was actively involved in the project's evolution and allowed for open discussions to address any concerns before progressing to the next iteration.

Requirements Captured

Functional Requirements

Data Input and Validation

The only time new questions and answers will be added into the system database is through the lecture. When new questions are added into the system the instructor should ensure the English question and SQL statement answer corresponds and behave as expected.

Data Retrieval and Display

- Students should only be able to complete an assignment after the opening date but before the closing date.
- Students should not be allowed to view the assignment answer before it has been closed and marked.
- Assignments marks are only released once every student has submitted and the assignment has been closed.
- The only data that will be visible to student at time they complete an assignment or practice test are English question and the data tables they are querying.

Error Handling and Logging

Great importance and focus were placed on data input and validation. Invalid data, which is the cause of many errors, was pre-empted and prevented from even being inputted at the presentation layer level. The possibility of errors was therefore rare, and the need to handle errors was few and far between.

Most errors that needed to be handled arose from interaction with the database. For example, when displaying the output from executing SQL statements on the database, some fields are empty null values that are received by Python as unusual data types. These data types could not be converted to and displayed as a string. To handle this error, a *try:* and *expect:* block was used, and an empty string was printed in the place of these unusual data types.

Error handling using a *try:* and *expect:* block was used again when executing a student's SQL statement answer from an assignment. A student may very likely enter an SQL statement that is invalid – incorrect syntax or spelling errors. If this happens, the error is caught, and the system returns the string *"error"* from the method that executes the statement. In the assignment feedback, the student is told if a specific statement of theirs was invalid.

User Training and Help Documentation

The system will be accompanied by a user manual which should be read and understood by the user before they interact with the system. This is pre-training that the student and instructor must walk through to ensure they understand the system prior to interaction with it for the first time. See *appendix B for user manual.*

Non-Functional Requirements

Performance

A specific system response time is not a requirement. However, for the sake of efficiency and user satisfaction, the system should take no longer than 10 seconds to evaluate SQL queries and generate feedback on a stable internet connection. It should upload and process lecturer information in under 5 seconds. The system must generate questions based on the inputted lecturer specifications in under 5 seconds per 100 students.

Security
To protect student data and information every system user is someone registered into the university with accredited student or staff number from the computer science department. Depending on the level of authority and individual has in the department they only will be granted access to data that pertains to their line of work and their interaction with the system to complete delegated department duties. Every user will then be required to have a password and username which will be used to determine how much system data they should be exposed to.

Usability Requirements
We have designed the user interface to be as simple as possible, making sure we only include necessary and important information on our screens, avoiding distractions and unnecessary complexity that would clutter users' focus. I our design we have employed various UI design principles to make sure we build an easy-to-use software that increase students' efficiency when completing assessments.

Clarity:
This system will be used by students while taking closed practical in the labs. We made sure to include descriptive button names and limit the information we show to a student to one question on a screen at a time only showing the tables they supposed to query, text box to written answers and the test button where they can test answer before submitting. All button names and content on screen is in English as it is the language of instruction in the institution. This is to allow ease system navigation allowing student to focus on what really matters under time constraints.

Consistency:
All buttons intended for the same functionality have the same color and size throughout the system and are only visible when the student needs to execute their functionality. This fosters familiarity which builds up as the user interacts with the system and builds ease of instruction execution increase productivity and reduce frustrations.

Feedback:
Each time a user executes an instruction that requires button click or any other user-system interaction they get a pop-up message on screen detailing the success or failure of their execution. The system then gives a discretion of the error and allows the user a chance to correct their deeds by either reentering information or restarting the execution from scratch depending on the depth of the error. This idea of instant feedback helps users understand the system's response to their actions in real time, so they can correct their deeds.

Simplicity:
Information shown on a screen only pertains to a single instruction execution to avoid unnecessary content that can overwhelm users. This leads to easy navigation and clarity to steps to be taken to complete a task at hand, and it gives progress satisfaction to the user.

Efficiency:
The instruction-task separation in our UI design increases user productivity, by reducing destruction ensuring they can accomplish their tasks quickly and with minimal effort.

Error Prevention:
System user interface includes descriptive labels, provides warning in terms of user input type expected, and offers undo options. This reduces the likelihood of user errors.

Compatibility:

The system is built so that it can be compatible with the UCT Vula student site that is used for academic activities across the institution.

User training:

The system is accompanied by a clear and comprehensive user manual detailing system navigation from different user perspectives. This gives an overview of the task completion step for all the tasks that can be accomplished using the system. Students also get to practice questions with assignment similar structure to allow system familiarity before going for the main task, to make sure they know their way around the system they allow encouraged to complete these exercises.

Availability:

The system should be available to students to at least up to the end of semester when they are taking the database course. During this period students should be able to use the system whenever they need to practice or complete an assignment as part of course work.

Use Case Narratives

Use Case: Generate SQL Assignments based on student numbers and specified criteria.

Actors: Instructor

Description:

- The use case begins when the lecturer logins using valid login credentials.
- The system checks that the login credentials are valid.
- The lecturer creates a new assignment.
- The lecturer inputs question-answer pairs and can select the distribution of easy, medium, and tough questions.
- The system then generates more examples given the lecturer's input. The system then creates an assignment for each student number in the class stored by the system.
- The system presents a message to the lecturer confirming that all students now have their respective assignments.

Alternate Paths:

- The lecturer attempts to login using an ID not recognized by the system. The lecturer is then presented with a message telling them to retry inputting their ID.
- The lecturer tries to input text not recognized by the system. The system then tells the lecturer that this text is not recognizable and must be re-entered.
- The lecturer attempts to input SQL statements with syntax errors. The system then warns the lecturer that this SQL statement doesn't execute and must be redone.
- The lecturer's SQL statements reference table names, and entries which are not present in the Classic Model's database. The system warns the lecturer, prompting them to redo it.

Pre-conditions:

- The lecturer must be logged in.
- The lecturer must have released the assignments for the class.

Post-Condition:

- The lecturer has finished generating assignments for the class and received a notification from the system confirming the completion.

Use Case: Take and Submit Assignment

Users: Student

Description:

- Use case begins when the student logins into the system using valid login credentials.
- The system checks that the login credentials are valid.
- The student begins the assignment.
- The student enters their respective SQL answers for the questions presented to them.
- The student submits the assignment.
- The system checks the student SQL output, comparing it to the model SQL output answers.
- The system grades the answers after comparing, and saves the grade to the database.

Alternate Paths:

- The student attempts to login using an ID not recognized by the system. The student is then presented with a message telling them to retry inputting their ID.
- The student tries to take the assignment before the lecturer has released them. The system then blocks the user from taking the assignment and tells the user that the lecturer has not released the assignment yet. The student stays on the home page.

Pre-conditions:

- The student has submitted the assignment and received a notification from the system confirming the submission.
- The student has been taken back to the home screen.
- The lecture has been taken to the home screen.

Use Case: View Grades

Actors: Student, Lecture

Description:

- Begins when either the student or lecturer logs into the system using valid credentials.
- The system verifies that either login credential is valid.
- The student views their grade and compares their answers to the model answers.
- The lecturer views the class's grades.
- The system presents the lecturer with the class average and other useful statistics which the lecturer can then use.

Alternate paths:

- The user attempts to login using an ID not recognized by the system. The user is then presented with a message telling them to retry inputting their ID.
- No students have submitted the assignment yet. If the lecturer tries to view grades when this is true, then the system informs them that no students have submitted the assignment yet. The lecturer stays on the home screen.

Pre-Conditions:

- The student has already submitted their assignment and answers.
- Students have submitted their assignments. So grades are present for the lecturer to view.

Post-conditions:

- The lecturer has exited to the home screen.
- The student has exited to the home screen.

Use Case: Assignment marking and grading
Actors: System
Description: • Begins when a student submits an assignment or practice exercise for grading. • The system stores student answers for the assignment in the database. • Then the system executes the submitted SQL statements and runs it against question answer stored in the database comparing outputs. • Then a grade is allocated per question and an overall assignment mark calculated. Alternate Path: • The student submits a blank answer or an invalid SQL statement. • The system has used an exception handler to deal with such cases and mark the student if the submitted answer cannot be executed . The description is saved in the feedback table. Pre-condition: • The student should have an assigned assignment or practice questions and have clicked the submit button. • The answers to the questions should be in the database for comparison.

Use Case: Practice Assignment
Actors: Student
Description: • It begins when the student clicks on the practice button. • The student enters the number of desired practice questions in the system prompt. • The system then generates questions for student practice. • The student is then taken into the practice screen with questions to answer. • The student then enters answers to questions and then submits. • The system then marks submission and returns feedback immediately. Pre-Conditions: • The student has already signed in on the system.

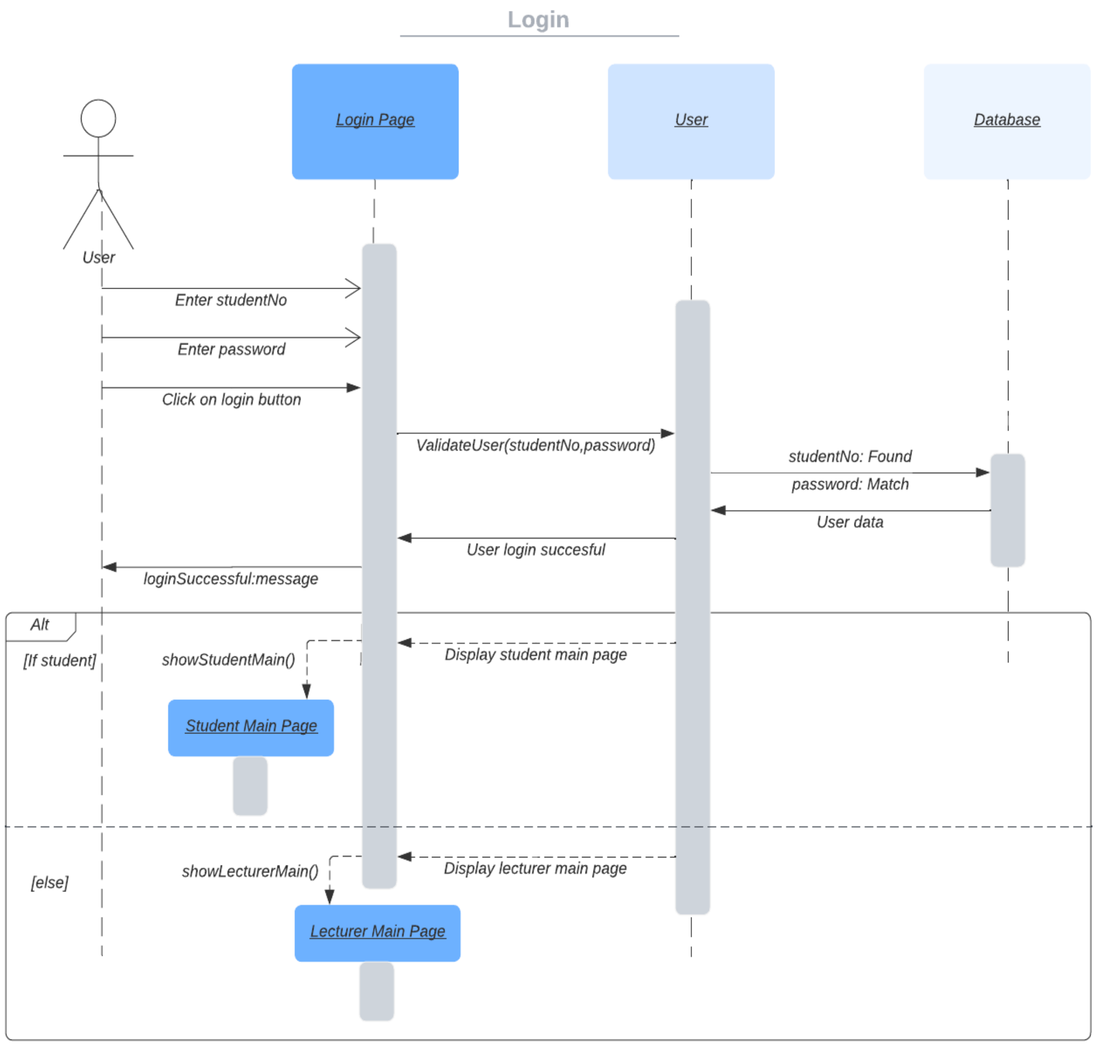
Use Case: Create Questions
Actor: System, Instructor
Description: • Begins when the lecture feeds example questions and answers to the system. • The system then creates additional questions, answers with varying difficulty levels, and stores them into the database. • The SQL statement created by the system is validated to check for errors if any error is detected, then a question will be rejected. Pre-condition: • The instructor has proved the example answers to be correct before inserted into the system.

Design Overview

TESTSQL empowers second-year computer science lectures with tools necessary to create and automate SQL assignments for their database courses and provides student with a structured tool to exercise their SQL skills and allow them to complete course assessment. Whereby:

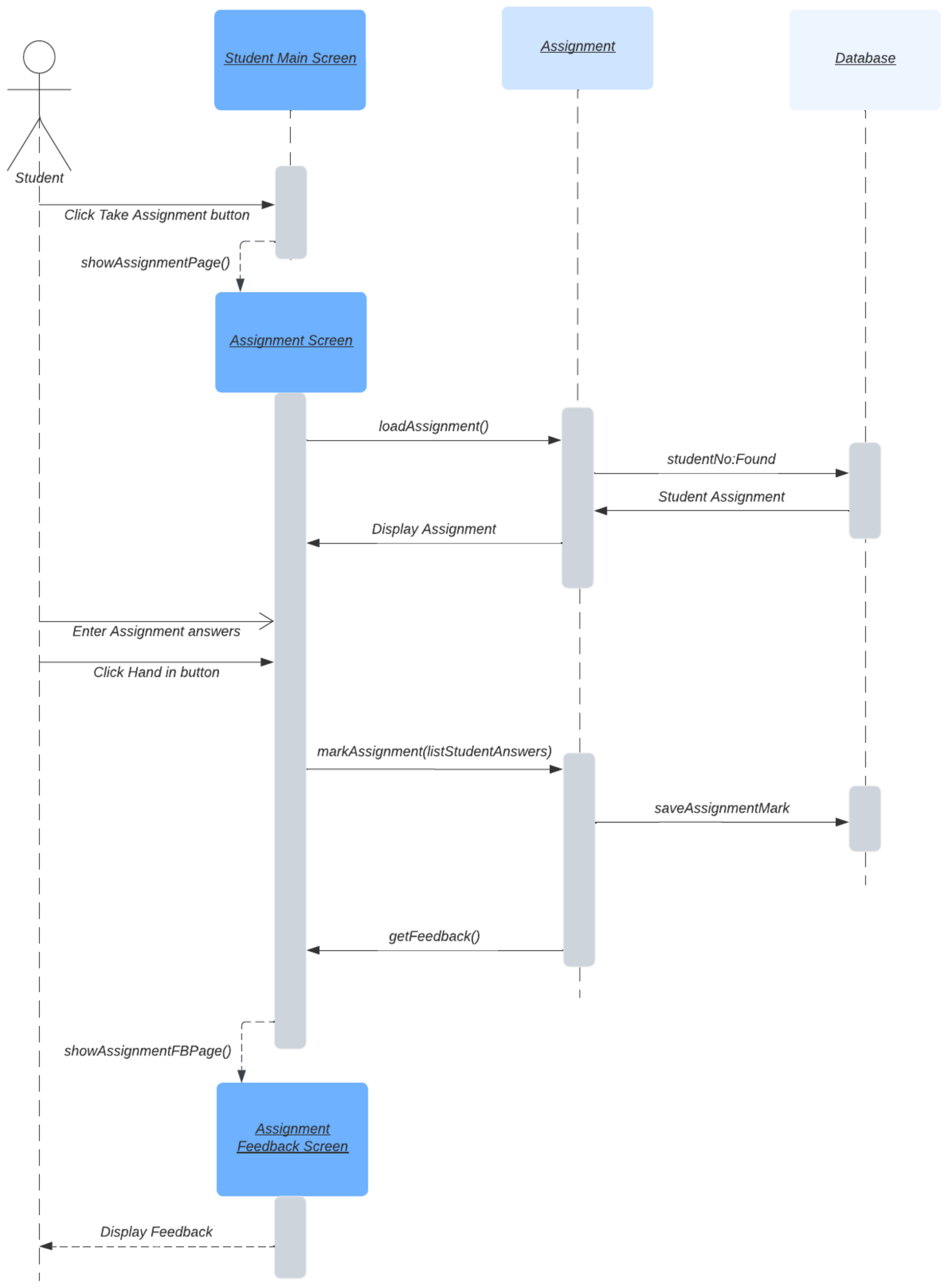
- The system user can sign into the system.
- The system allows lectures to tailor assessments based on desired criteria.
- With the aid of an efficient database, the system generates unique assignments for each student, ensuring fairness.
- Students can practice SQL with tailored questions that meet desired criteria.
- Students can complete class assessments which have been allocated by the instructor.
- The lecture and students can view grades of previous assignment.
- The system includes an automatic grading component that evaluates and provides feedback to student submissions.

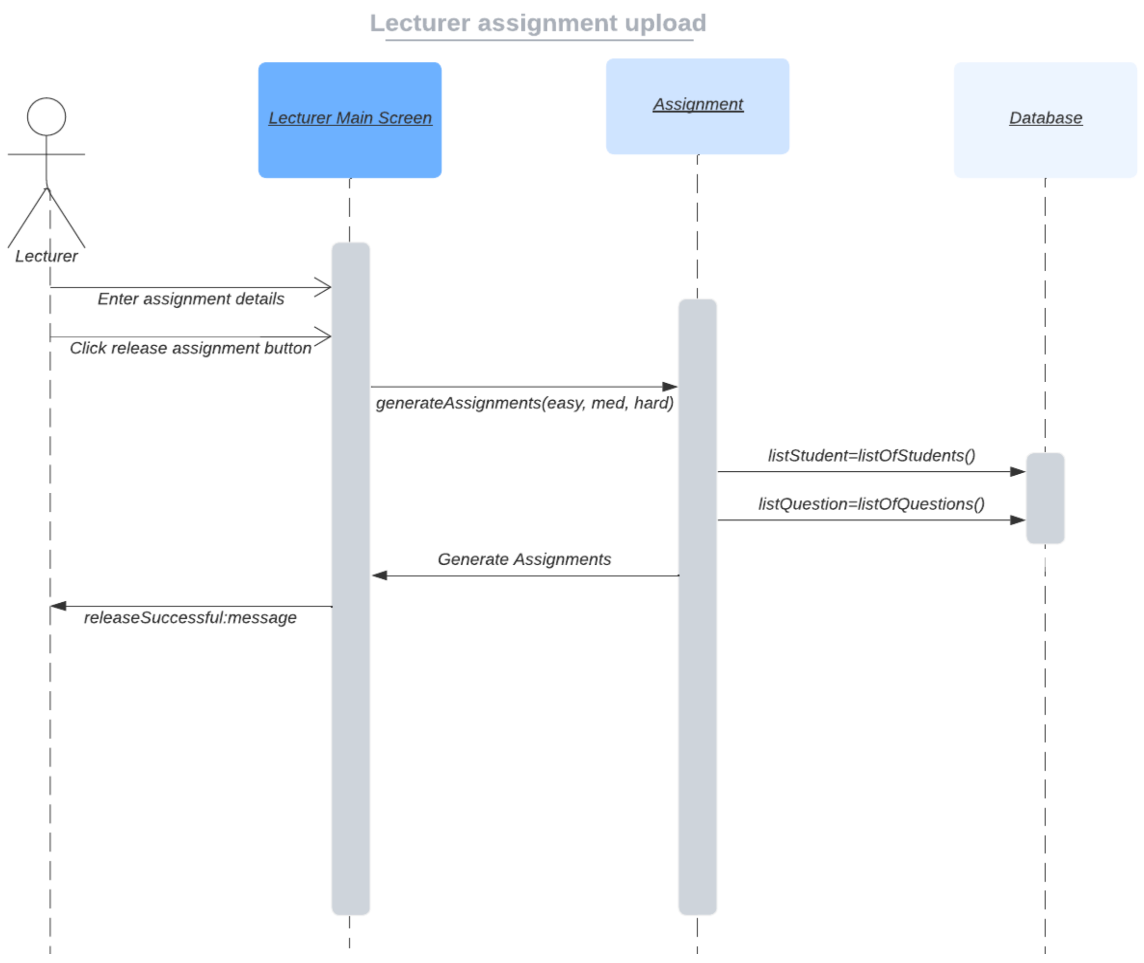
Sequence Diagram



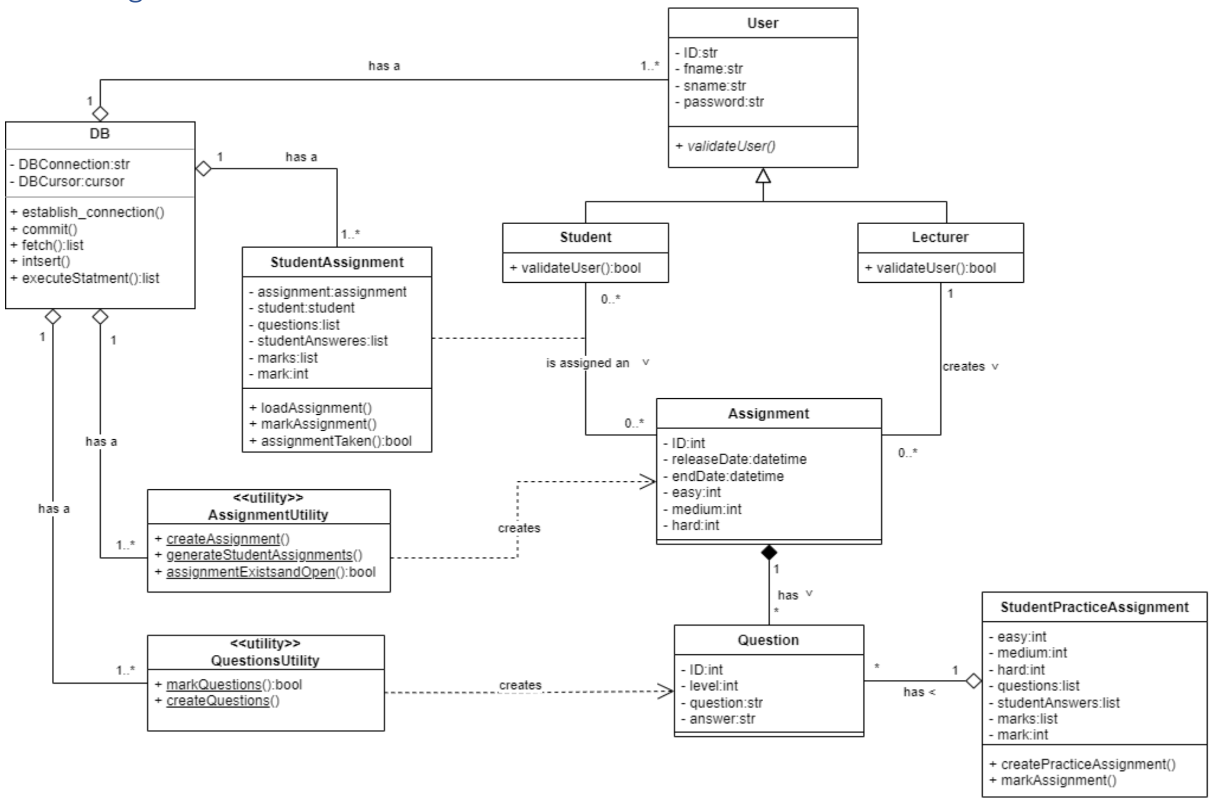
Take Assignment

Kane Gibson | September 18, 2023




Lecturer assignment upload
Lecturer Main Screen
Assignment
Database
Lecturer
Enter assignment details
Click release assignment button
generateAssignments(easy, med, hard)
listStudent=listOfStudents()
listQuestion=listOfQuestions()
Generate Assignments
releaseSuccessful:message

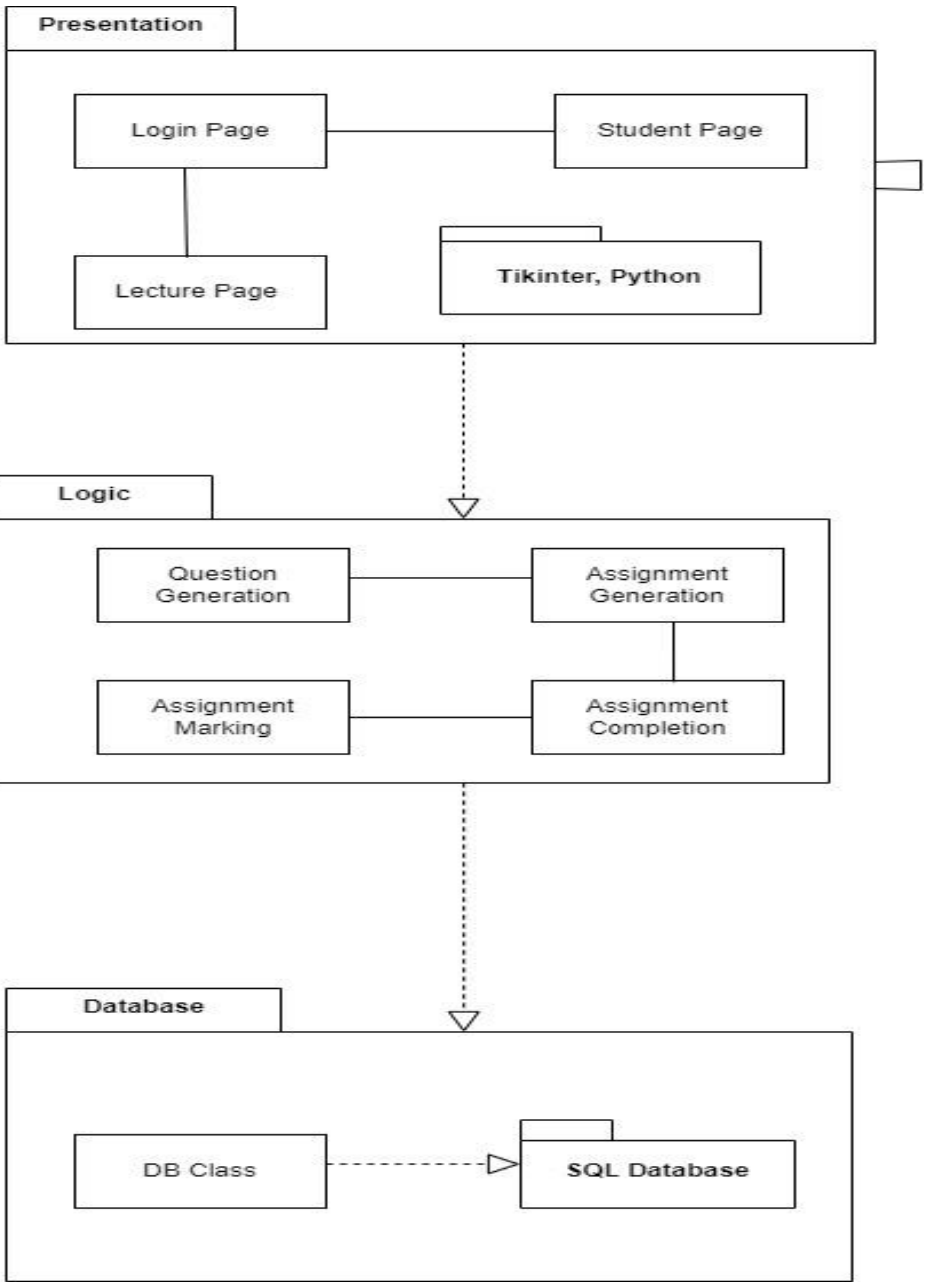
Class Diagram



System Architecture

The system functionality includes interaction of different system entities/components such as the user interface for user system interaction, classes within the system for system logic implementation, and the database for data storage. Thus, our system architecture design employs a layered architecture, where we separate our system in three layers which are distinguished by their functionality one layer is the user interface, logic /business layer, and database layer. Each layer is a module that consists of classes pertaining to the functionality dedicated for that layer, this promotes High Cohesion.

Architectural Diagram



Implementation

Presentation Layer

The presentation layer had the designated function of providing the GUI for the user to interact with. The presentation layer made use of the logic layer, which provided the necessary backend functions and classes to the presentation layer.

LoginForm:

The LoginForm class makes use of the tkinter library to provide a user-friendly graphical interface for users. The user can use this interface to input their login credentials (their user ID and password). The class handles user input validation and interaction, displaying messages to handle incorrect details being entered and messages greeting the user by their first name if the correct inputs were entered. This class is the first interaction which the user will have with the program. The class will then take the user to either the LecturerHomeForm class or the StudentHomeForm class depending on which details were inputted.

LecturerHomeForm:

If the login details inputted on the LoginForm were for a lecturer, then the user will be taken to this class. This class also makes use of the tkinter library. This class is responsible for providing the GUI for the lecturer home page. The lecturer home page allows for the lecturer to perform various tasks in relation to the creation of assignments and assignment questions and viewing grades. The form also dynamically changes the state of the 'Create Assignment' button depending on whether an assignment has already been created or not.

StudentHomeForm:

If the login details inputted on the LoginForm were for a lecturer, then the user will be taken to this class. This class also makes use of the tkinter library. This class provides the GUI for the student home page. The student home page provides the student with multiple functions , such as taking an assignment or practice assignment, and viewing their grades. Just like with the lecturer home form the 'Take assignment' button changes its state dynamically depending on whether an assignment has been created for the student ID which was just used to login.

CreateAssignmentForm:

The CreateAssignmentForm makes use of the tkinter library to create a user-friendly GUI for users who are lecturers. This class has the sole responsibility of providing a GUI in which the user can create assignments for every student in the database. This GUI provides 3 inputs where the user can specify how many easy, medium, and hard questions will be included in the assignment. The spinboxes used to capture the number of easy, medium, and hard questions have restrictions on them so that the user can input a number higher than the total number of questions in that category. For example, if there are 13 easy questions in the easy category, the user cannot input a number in the spinbox which is higher than 13. The class then takes the release date input and closure date input. Once all the specified inputs have been completed the user then clicks on the 'Submit Assignment' button, once they confirm the pop-up message, they are then taken back to the LecturerHomeForm class.

ViewGradesForm:

If the user clicked on the 'View Grades' button in the LecturerHomeForm page, they will then be taken to this class. This class makes use of the tkinter library as well, to provide a GUI for the user to interact with the program. This class has the sole responsibility of showing the lecturer the grades of the class and what each student achieved for the assignment. This class is dynamic and shows the updated table every time a new grade has been entered into the database.

TakeAssignmentForm:

This class provides a GUI in which the student can now take and submit assignments. This GUI was created using tkinter. The GUI displays all the questions the student was allocated for their assignment along with input fields for them to input their answers and buttons which would run their SQL statements and display the outputs in a text field below. The GUI is dynamic and uses iteration to display the questions, input fields, run buttons and output fields multiple times. A scrollbar is then implemented on the side to ensure that all the content can be accessed. The student then submits the assignment and is taken back to the StudentHomeForm.

AssignmentFeedbackForm:

The AssignmentFeedbackForm class provides a GUI in which the user(student) can view and receive feedback on their submitted assignment. This class makes use of tkinter to provide the GUI. The class displays the heading along with your overall mark at the top of the screen. The class makes use of a canvas, which allows the user to scroll through the screen. After the heading, each question is displayed along with your inputted answer alongside the model answers. Once the student is done looking through their feedback, they can exit to the student home screen.

TakePracticeAssignmentForm:

This class provides a GUI in which the student can now take and submit practice assignments. This GUI was created using tkinter. The GUI displays all the questions the student was allocated for their practice assignment along with input fields for them to input their answers and buttons which would run their SQL statements and display the outputs in a textfield below. The GUI is dynamic and uses iteration to display the questions, input fields, run buttons and output fields multiple times. A scrollbar is then implemented on the side to ensure that all the content can be accessed. The student then submits the assignment and is taken back to the StudentHomeForm.

PracticeAssignmentFeedbackForm:

The PracticeAssignmentFeedbackForm class provides a GUI in which the user(student) can view and receive feedback on their submitted practice assignment. This class makes use of tkinter to provide the GUI. The class displays the heading along with your overall mark at the top of the screen. The class makes use of a canvas, which allows the user to scroll through the screen. After the heading, each question is displayed along with your inputted answer alongside the model answers. Once the student is done looking through their feedback, they can exit to the student home screen.

AutoGeneration:

This class provides an interface for the lecturer to auto-generate question and answer combinations. This class makes use of tkinter to provide a GUI where the user will be allowed to create combinations and add them to the table. The class allows the user to input 2 strings which can then be used to transform them into multiple variations of the original strings. The GUI has dynamic states, to make sure that the user follows the correct procedure when creating the combinations. By making some buttons and input text boxes unavailable for the user to use, the user is then required to follow specific steps to create their combinations. The class makes use of multiple arrays to store both the answer combinations and the question combinations. The class also gives the user the ability to delete some combinations from the list. Once the user has clicked the 'Create and Exit' button the combinations will be added to the database with their specified Level and newly created question IDs. The user is then taken back to the LecturerHomeScreen.

SingleAddition:

The class provides a GUI in which the user can add questions one at a time with alternating difficulty levels. The class makes use of tkinter to provide the GUI. The class has 2 text input boxes, where the user can enter the question and respective SQL answer, and a spin-edit box where the user can specify the difficulty of the question. The class makes use of arrays to store the questions, its answer, and its difficulty level. When the user adds the combination to the list, the class informs the user of the addition using a message box. Once the user is done adding their combinations, they can then submit them to the database and exit to the home screen.

Logic Layer

User Class:

An abstract base class representing a user with common attributes such as ID, first name, surname, and password. It provides a template for subclasses (Lecturer and Student) to implement user validation. Only accessor methods, used to set attribute values, exist in this class.

Lecture Class:

A subclass of *User* representing lecturers. It inherits user attributes and provides a method to validate lecturer credentials against the database.

Student Class:

A subclass of *User* representing students. It inherits user attributes and provides a method to validate student credentials against the database.

Question:

This class represents an SQL question with attributes including an ID, difficulty level, the question text, and the correct answer. The class has getter methods for each of these attributes.

Methods:

- getID() - Returns the question's ID.
- getQuestion() - Returns the SQL question.
- getDifficulty() - Returns the difficulty level.
- getAnswer() - Returns the correct answer.

Assignment:

This class represents an assignment with attributes including an ID, release and end dates, and the number of questions at each difficulty level (easy, medium, hard). This class must not be confused with an assignment that is unique and linked to a specific student. Rather, it is a broad assignment that a lecturer can release that specify the details of the assignment.

Methods:

- getID(): Returns the assignment's ID.
- getReleaseDate(): Returns the release date.
- getEndDate(): Returns the end date.
- getEasy(), getMedium(), getHard(): Return the number of questions for each difficulty level.

StudentAssignment:

Represents an assignment for students, including the associated Assignment and Student object, a list of the questions in the assignment, the student answers, marks for each question, and the total mark. This class links an assignment and a student. This can be seen as a unique specifically linked to an individual student.

Methods:

- getQuestions(): Returns the list of questions.
- setStudentAnswers(studentAnswers): Sets the student's answers.
- loadAssignment(): Loads up an existing student assignment. It fills the list questions, as well as the student answers, marks for each question, and total mark if the assignment has been taken by the student.
- markAssignment(): Evaluates student answers and calculates the marks. Saves answers and marks to the database
- assignmentTaken(): Checks if the assignment has been taken by the student.

StudentPracticeAssignment:

Represents a practice assignment for students. Unlike the **StudentAssignment** class, it has no Assignment and Student object, because it is a once of practice assignment that saves no data. Attributes include a list of the questions in the assignment, the student answers, marks for each question, the total mark, and the number of easy, medium, and hard questions.

Methods:

- getQuestions(): Returns the list of questions.
- setStudentAnswers(studentAnswers): Sets the student's answers.
- createPracticeAssignment(): Generates random practice assignments based on difficulty levels.
- markAssignment(): Evaluates student answers and calculates the marks. Does not save anything to the database

AssignmentUtility (Static Class):

A static utility class containing methods for creating and managing assignments. The methods can be seen as helper methods that utilize the basic classes like **Assignment** and add functionality. The class has no attributes.

Methods:

- createAssignment(releaseDate, endDate, easy, medium, hard): Creates and adds an assignment to the database.
- generateStudentAssignments(assignment): Generates unique assignments for each student and adds them to the database.
- noTotalQuestions(level): Returns the total number of questions at a given difficulty level.
- getNextAssignmentID(): Gets the next available assignment ID from the database.
- assignmentExistsAndOpen(): Checks if any assignments exist and if they are open for taking.

QuestionsUtility (Static Class):

A static utility class containing methods for creating and managing questions. The methods can be seen as helper methods that utilize the basic classes like **Question** and add functionality. The class has no attributes.

Methods:

- markQuestion(correctAnswer, studentAnswer): Marks a question based on the correct and student's answers.

- getLastQuestion(level): Gets the last question of a given difficulty level from the database.
- createQuestions(id, level, question, answer): Adds new questions to the database.
- getTableName(tablename): Gets column names of a table in the database.
- getTables(): Gets the names of tables in the database.

Database Layer

DB Class:

The database DB class is responsible for all the system databases related interactions. It is implemented to follow the singleton design pattern, thus in the system there is only one instance of such a class and is used for all database related functionalities. This class has three methods designed used for different database interactions as follows:

- Fetch (): This function accepts as an argument a string table name and returns the data for that specific table.
- Insert (): This function accepts as an argument a table name, the data to be added into the database, it is used when a new data entry needs to be added into the database.
- ExecuteStatement(): This function accepts as an argument an SQL statement and use the execute function to cursor function to execute the statement and return data.

Every class that interacts with the database in the system has an instance of this class as it is the only way to interact with the DB.

Program Validation and Verification

Table 1: Summary Testing Plan.

Process	Technique
1. Class testing: testing the methods and state behaviours of the classes	Random, partition and white-box testing
2. Integration testing: testing the interaction of sets of classes	Random, boundary and behavioural testing based on given requirements.
3. Validation testing: testing whether the client's requirements are satisfied	Use-case based black box and acceptance test
4. System testing: testing the behaviour of the system as part of a larger environment.	Recovery, security, stress and performance tests

Class Testing:

We began writing classes with their respective methods first. Once these classes were written and had full individual functionality, we could then begin testing them. The designated tester in the group began testing each class individually to ensure that each was fully functional, along with their methods and state behaviors. The tester conducted random, partition and white-box tests. The random and partition tests were used to evaluate the expected behavior of each classes' methods with boundary and expected input data. The white-box testing provided the tester with all the design documentation and knowledge of the internal functioning of each class. This ensured all classes functioned correctly.

Integration Testing:

Once all the different classes were written and fully functioning individually, then integration testing commenced to make sure that all these individual classes could be integrated with each other to form the complete system and to assess the interaction between the different classes. Behavioral and boundary tests were completed on various parts of the system to test their individual success and readiness to be integrated into the complete system. The behavioral tests ensured that the integrated parts behaved as expected. Random testing ensured that the integration of the different classes did not result in any unexpected behavior. The random testing was especially effective in uncovering unexpected issues which occurred during integration.

Validation Testing:

Once integration testing was complete validation testing began immediately. After our first demo session with Prof. Berman, we made a list of all recommended changes and features she wanted included in the final product. The validation testing was then conducted to verify that our final system satisfied all the requirements. Use-case based black box testing was used to simulate real-world scenarios which the user will be in. The acceptance tests were conducted to confirm that the system met all acceptance criteria. Along with the validation our team scheduled a follow-up meeting with Prof. Berman, to ask for further feedback after our program had incorporated all the previous recommendations. This meeting also ensured that user acceptance tests would align with Prof. Berman's expectations.

System Testing:

The final stage of testing was system testing. System testing would assess our program's behavior in a larger environment. Security testing was conducted to check for vulnerabilities in our program. Stress testing examined how our system handled heavy loads. This was tested by creating large number of assignments for very large classes and ensuring that all assignments were added to the database. It also tested if a large quantity of auto-generated questions was generated then the system could handle it and add them to the database. The performance testing assessed the response times of our system. Recovery testing was used lastly to ensure that our system would be able to recover from failures.

Summary of tests carried out:

To ensure that our application was, several tests were run to ensure that a new user would have no issues or misconceptions when using the application. Several tests were conducted with each test the test data used was testing different test conditions. For the login form, we tested to see that only valid inputs would allow the user to proceed to the respective home screen depending on their role. For the take assignment button on the student home screen and the create assignment button on the lecturer home screen, the test data used for these showed that new assignments can only be created if no assignment is present in the database and an assignment can only be taken if there is an assignment in the database. For the practice assignment and assignment screen we used test data which checked to see that only valid inputs could be run using the run buttons and only the correct answers would be marked correct (or answers which produced the same outputs as the model answers). The auto generation and single addition screens were tested with test data which ensured that user input was controlled and structured. The view grades form was tested to show that every time a new mark had been added to the database then the screen would reflect this change. The feedback screens in the student class were then tested with test data which ensured that correct answers were marked correctly, and all incorrect answers were marked incorrectly.

Table 2: Summary of Tests Carried Out.

Data Set and reason for its choice	Test Cases		
	Normal Functioning	Extreme boundary cases	Invalid Data (program should not crash)
Login Form: Invalid inputs (Appendix A: row number 1)	Passed	n/a	Passed
Login Form: Valid inputs (Appendix A: row number 2)	Passed	n/a	

Lecturer:			
Home Screen (Appendix A: row number 3)	Passed	n/a	
View grades test (Appendix A: row number 4)	Passed	n/a	
Auto-generation test (Appendix A: row number 5)	Passed	Passed	
Assignment creation test (Appendix A: row number 18)	Passed	Passed	
Single question addition test (Appendix A: row number 6)	Passed	Passed	
Student:			
Home screen test (Appendix A: row number 7)	Passed	n/a	
Take assignment test (Appendix A: row number 8, 9 & 16)	Passed	n/a	Passed
Take practice assignment test	Passed	n/a	Passed

(Appendix A: row number 17)			
Practice assignment marking test (Appendix A: row number 12 & 13)	Passed	n/a	Passed
Assignment marking test (Appendix A: row number 10 & 11)	Passed	n/a	Passed
Practice assignment feedback test (Appendix A: row number 15)	Passed	n/a	
Assignment feedback test (Appendix A: row number 14)	Passed	n/a	

Conclusion

We the developing team feel that after the completion of this project we have delivered an application which meets and exceeds all the sponsor's requirements. We tested our application to ensure that all these requirements were met, using various black-box and white-box tests. We have developed an application which can be used to design and create SQL assignments/practice assignments and allow students to take these newly created assignments/practice assignments. Our system was broken up into three well-structured layers; the presentation layer, logic layer and database layer. This made our code easier to understand, easier to modify and easier to extend functionality across the different classes. By having such a modular design, it allowed for easier integration and collaboration amongst our team. An extra consideration put in place was to ensure that our program could function with any database the lecturer specifies. At the end of our development, we were pleased to have created a robust and complete system which meets our sponsor's requirements and expectations.

Appendix

Appendix A

No	Tests	Test Data 1	Test Data 2	Test Data 3
1	Login Form: Invalid inputs	User ID=1 and user password=hello12	User ID=0 and user password=hello1	User ID='' and user password=''
2	Login Form: Valid inputs	User ID=1 and user password=hello1	User ID=2 and user password=bye12	User ID=8 and user password=Sunday
3	Lecturer Home Form: Create assignment button test	Assignment table is empty	Assignment table has an assignment and it has not been released	An assignment has just been created
4	View grades form: different table states	No students have taken the assignment yet	Students have taken the assignment	-
5	Auto-generate screen: different versions of the auto generated combinations	Combinations generated and none were deleted	Combinations and only one deletion list was inputted	Multiple deletion lists used on the original combination list
6	Single question addition: different states of completion	No combinations have been created	Only a singular combination has been created	Over 20 combinations were created
7	Student Home screen: Take assignment button states	Assignment table is empty	Assignment table has an assignment and it has not been released	An assignment has just been created
8	Student takes assignment: valid inputs run	Due to the questions being random for the assignments the question's answer was looked up in the database to make sure a correct answer was inputted		
9	Student Take assignment: invalid inputs run	Because questions are random for the assignments, the question's answer		

		was looked up in the database to make sure an incorrect answer was inputted		
10	Student assignment marking: Invalid inputs	Due to the questions being random for the assignments the question’s answer was looked up in the database to make sure an incorrect answer was inputted		
11	Student assignment marking: Valid Inputs	Due to the questions being random for the assignments the question’s answer was looked up in the database and then inputted		
12	Student practice assignment marking: Valid Input	Due to the questions being random for the assignments the question’s answer was looked up in the database to make sure an incorrect answer was inputted		
13	Student practice assignment marking: Invalid input	Due to the questions being random for the assignments the question’s answer was looked up in the database to make sure an incorrect answer was inputted		

14	Student assignment feedback	A prerequisite would be that the student had submitted an assignment with multiple correct and incorrect answers, to see that the marker awarded marks correctly		
15	Student practice assignment feedback:	Null answer submitted in practice assignment	Incorrect answer submitted in practice assignment	Correct answer submitted in practice assignment
16	Student taking assignment and clicks run button	Null answer being inputted	Incorrect answer inputted	Correct answer inputted
17	Student taking practice assignment and clicks run button	Null answer being inputted	Incorrect answer inputted	Correct answer inputted
18	Create assignment	Incorrect input for release date field or end date field	Null input for release date field or end date field	Correct input for release date field and end date field

Appendix B

See user manual uploaded as separate document.