Executive Summary **1**
Chapter 1: Introduction **2**
Background and Motivation 2
Problem Statement 2
Objectives 3
Expected Outcomes 3
Chapter 2: Literature Review **4**
Chapter 3: Software Design & Modelling **5**
Primary: Model View Controller (MVC) Pattern 10
SOLID Principle 12
S - Single Responsibility Principle (SRP) 12
O - Open/Closed Principle (OCP) 12
L - Liskov Substitution Principle (LSP) 13
I - Interface Segregation Principle (ISP) 13
D - Dependency Inversion Principle (DIP) 13
Inversion of Control (IoC) 13
Chapter 4: Construction & Unit Testing **14**
Core Logic 14
Service Layers: 14
Creational Patterns for Scalability: 14
Data Persistence (Repository Pattern): 15
Unit Testing 15
Test Strategy 15
Question Bank 16
Student Database 17
True or False Question 18
Chapter 5: Integration & Application **19**
GUI Development 19
Authentication Interfaces 19
Home Interfaces 20
Profile Interfaces 21
Quiz Selection Interfaces 22
Quiz Question Interfaces 23
Quiz Review Interfaces 24
Quiz History Interfaces 25
API Integration 26
System Testing 27
Chapter 6: Conclusion & Professional Practices **29**
Demonstration Video 29
Reflection 29
References **30**

Appendices.....31
Appendix 1 - Source Code Github Repository.....31
Appendix 2 - Demonstration Video.....31
Appendix 2 - Unit Test Report..... 32
Test Case Examples.....2

Executive Summary

In this project, the development of an Educational Testing System has been proposed to overcome the shortcomings of conventional testing methods, which mostly depend on final results. To enhance the learning outcome, the system will ensure that there is dynamism in the generation of quizzes, assessment of the students' reply and feedback that is meaningful to identify areas of strength and weaknesses in concepts.

The solution is developed as a Java based object-oriented application that applies Model-View-Controller (MVC) architecture to provide a clear separation between the application logic, user interface and data management. Its key functions are user authentication, randomized creation of quizzes based on a question pool, automated evaluation of answers, score calculation and delivery of feedback. Design patterns like the Factory Pattern are implemented on enhancing modularity as well as maintaining it.

The JUnit-based unit and system testing can ensure that the system is correct and reliable. On the whole, the project helps to achieve SDG 4: Quality Education by offering a convenient, well-organized, and efficient evaluation platform, which improves the knowledge level beyond the numerical indicators.

Chapter 1: Introduction

Background and Motivation

The dynamically evolving society combined with the rapid shifting of technologies causes the need for a change of landscape in the educational sector. Supported by the 4th United Nations Sustainable Development Goals (SDG) which states the demand for an inclusive and equitable education, technologies are often overlooked in terms of the potential they could bring to the table for the purpose of an improved educational system.

For decades, formal education has often been characterized with the typical, linear-style learning processes. However, With the unprecedented growth of technology and digital space, static, fixed-curriculums may no longer be a viable and optimal implementation for the educational sector. The emergence of new publicly-facing technologies such as Artificial Intelligence (AI), Machine Learning (ML), and advanced Data Visualization has created an environment in which formal education is not anymore the only path to knowledge.

Problem Statement

Until now, traditional tests have never been able to cover one of the most important and underlying factors which influences a student's performance, their fundamentals. While the generated score does show a raw, uninfluenced indicator of a student's capability, more often than not, it instead hinders the performance of underperforming students. A test could reveal a student's weakness in intermediate mathematical operation questions, although not revealing the student's lack of understanding in PEMDAS which is the fundamental cause. Most tests done in the traditional way would pinpoint their mistakes, but not the underlying, overlooked cause.

Generative Artificial Intelligence (AIs) had introduced a new unprecedented access to information, even counteracting the highly-dynamic, evolving society. Fortunately, such emerging technologies could serve as a viable solution to perfect the aforementioned gaps found in traditional testing systems. The capability for these technologies to consume dynamically-driven, heavily-varying data reflects the nature of educational ecosystems itself, where no students are the same with another. While no guarantees could be given to the true results, the implementation of these technologies into existing processes may provide an opportunity for the fundamental gaps of the educational landscape to be filled.

Objectives

- To create a user-centered and easy-to-use educational testing system in order to promote accessibility and inclusivity of the app for a wide range of audience.
- To create a dynamically adapting educational testing system which caters to the unique traits of each user, pinpointing their weaknesses and strengths.
- To create a testing system which highlights potential key misunderstandings and rooms for improvement, eliminating the fundamental (conceptual) issue left uncovered by traditional testing methods

Expected Outcomes

- Deliver an increase in user interaction, engagement, and individual relevancy through our personalized educational testing system.
- Deliver a system which successfully promotes increased accessibility and ease of use for individuals to identify their conceptual gaps/problems.

Chapter 2: Literature Review

In a discovery revealed by research (MATSH, 2024), other than a 12% increase in attendance rate, the implementation of Personalized Learning, which includes adaptive tests, also improves student retention rates by up to 15% and a remarkable 30% increase for test performances. Additionally, in the context of emotional cognitive aspects, it is also revealed that student engagements increased by 20% and significantly, 45% more students mentioned being motivated compared to traditional learning methods.

In terms of the student preferences itself, another research (Cătălina Mărcuță, 2024) has revealed that 67% of students prefer platforms with a real-time, personalized, and direct feedback system. In the end, the implementation of such systems had generated an outstanding effect of 25% improved retention rates, 73% more motivation, and a major 30% increase in learning outcome. Moreover, for low-achieving students, a dramatic shift of 13.35% overall performance increase compared to the 10.22% decrease when individual feedbacks are not given was also identified, along with a higher 36-40% increase of overall performance for mid-achieving students (Akpinar & Özalp, 2023).

Quality Education, the 4th Sustainable Development Goals (SDG 4) does not consist of only finding the correct, static formula (e.g. curriculums) to increase student performances. It also covers inclusivity, accessibility, and ease of use which has been found to significantly affect learning performance through boosted relevance with the student. As such however, inclusivity itself would not take into effect without the system being able to identify the unique and highly dynamic characteristics of each student individual.

The aforementioned research had shown how a relevant, personalized system which uniquely caters to individuals could potentially improve learning processes and eventually the outcome by a significant margin. Identifying, modifying, and catering a relevant “sweet spot” for each individual user does not only affect their outcomes, but also to further reinforce Quality Education, as mentioned by SDG4. By delivering a personalized experience, learning processes could be strengthened as an actual educative process, rather than an activity done to achieve scores and/or performance metrics by any means possible.

Chapter 3: Software Design & Modelling

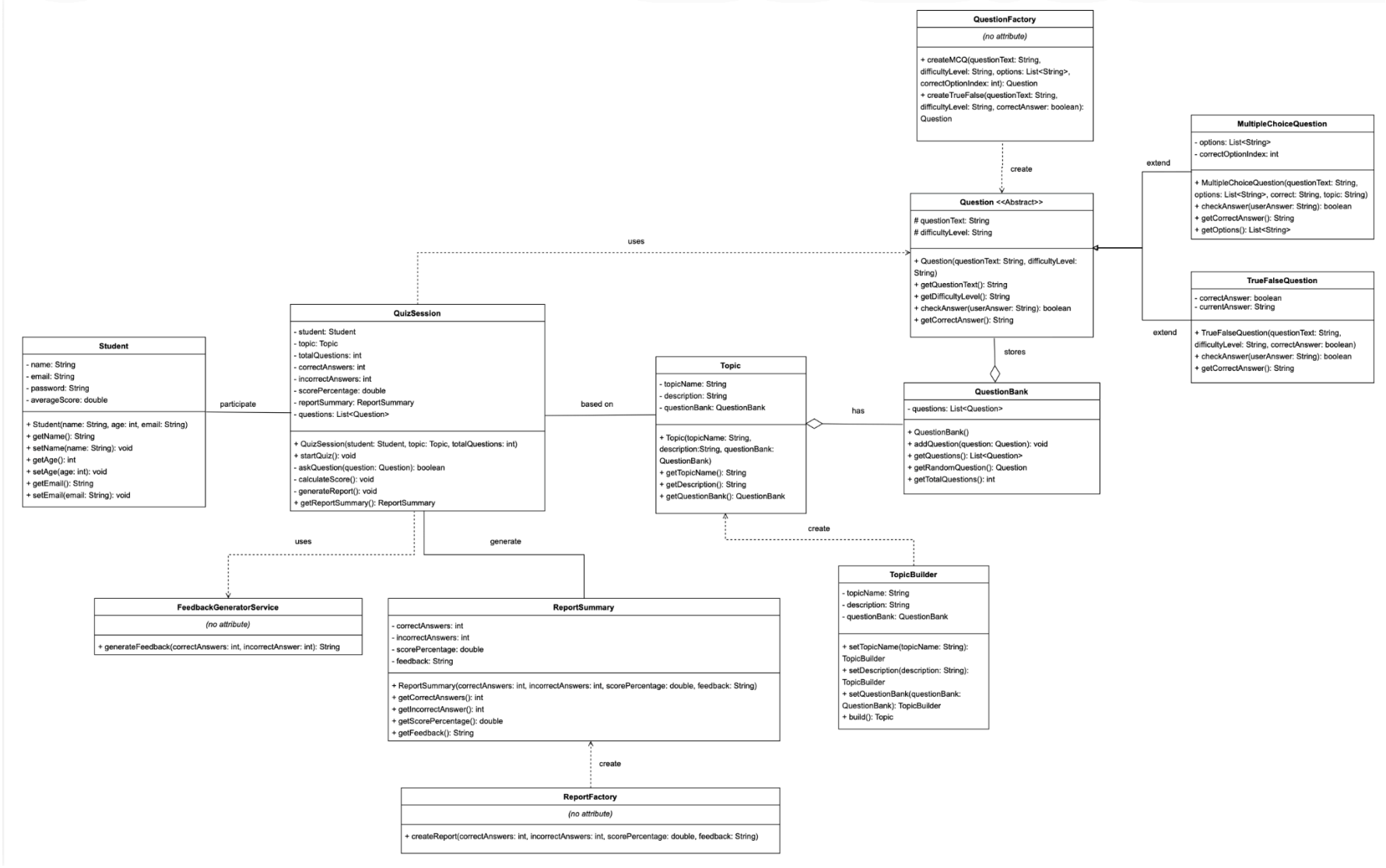
Due to the nature of our project which involves delegation of tasks and separate responsibility to minimize errors, the presented list serves as the primary high-level overview for the features that shall be implemented in the software:

1. **Authentication and Profile System**
    1.1. User Registration (Input Email, Username, and Password)
    1.2. User Log In (Input Email & Password)
    1.3. User Log Out
    1.4. Profile Page
        1.3.1. Show Username
        1.3.2. Show User Statistics
        1.3.3. Show Quiz History

2. **Quiz System**
    2.1. Quiz Initiation
        2.1.1. Show Available Topics & Short Description
        2.1.2. Select and Start Topic
    2.2. Quiz Runtime
        2.2.1. Show All Questions & Answer Field
        2.2.2. Select Answers
        2.2.3. Submit/Finish Quiz
    2.3. Quiz Evaluation
        2.3.1. Show Final Score
        2.3.2. Show Feedback

**The mentioned overview only covers what shall be expected from the client-side (GUI Development). The explicit work packages and implementation that needs to be done within the software (e.g. Score Algorithm, External API for Feedback) are covered on Chapter 4.*

Architecture - UML Class Diagram



UML Class Diagram Description

The description are categorised into available relationship, as follows:

I. Dependency Relationship

Dependency relationships are used in the system when a class uses another class temporarily to perform a task. In these cases, the dependent class does not store the other class as a long-term attribute and only interacts with it when needed.

The ***QuizSession*** class depends on ***Question*** because it retrieves and uses questions during the quiz process to display them to the student and check answers. The questions are not owned or permanently stored by the quiz session; instead, they are used only while the quiz is running.

The ***QuizSession*** class also depends on ***FeedbackGeneratorService*** to generate feedback after the quiz is completed. This service provides feedback logic based on the quiz results and does not store any quiz session data. The quiz session simply calls the service methods when feedback is required.

The ***QuestionFactory*** class depends on ***Question*** because it is responsible for creating different types of question objects, which are multiple-choice and true/false questions. The factory encapsulates the object creation logic and returns question instances without maintaining any reference to them. This ensures flexibility in question creation while keeping the system loosely coupled.

Similarly, the ***ReportFactory*** class depends on ***ReportSummary*** because it creates report summary objects after the quiz ends. The factory focuses only on constructing the report object using provided data and does not manage or store the report itself.

The ***TopicBuilder*** class depends on ***Topic*** because it constructs topic objects step by step before returning the final instance. The builder simplifies the creation of complex topic objects by allowing attributes to be set incrementally. The topic itself does not depend on the builder, confirming that the relationship is one-directional.

II. Inheritance Relationship

Inheritance is used in the system to represent an "is-a" relationship, where subclasses share common attributes and behaviors from a superclass while providing their own specific implementations.

The ***MultipleChoiceQuestion*** class inherits from the abstract ***Question*** class because it is a specific type of question that follows the same basic structure, such as having question text and difficulty level. However, it implements its own logic for checking answers and handling options.

Similarly, the ***TrueFalseQuestion*** class inherits from ***Question*** because it represents another specific type of question with a different answer format. By inheriting from the abstract question class, both question types can be treated uniformly during the quiz process while still maintaining their unique behaviors.

III. Aggregation Relationship

Aggregation relationships are used when a class contains or manages other objects, but does not strictly own their lifecycle. This means the contained objects can exist independently of the container.

The ***Topic*** class aggregates ***QuestionBank*** because a topic groups and manages its questions through a question bank. The question bank is provided to the topic and is not created internally, which allows it to exist independently. This makes aggregation more appropriate than composition, as the topic does not fully control the lifecycle of the question bank.

Similarly, ***QuestionBank*** aggregates ***Question*** because it maintains a collection of question objects that are used in quizzes. Questions are created separately and added to the question bank when needed. Since questions can exist independently and are not destroyed when the question bank is removed.

IV. Association Relationship

Association relationships are used in the system when classes are linked to each other for collaboration, but can exist independently. These relationships represent logical connections during the quiz process without strong ownership.

The ***QuizSession*** class is associated with ***Student*** because each quiz session is conducted for a specific student. The quiz session needs the student's information to track participation and results, but the student object exists independently and can take multiple quiz sessions over time.

The ***QuizSession*** class is also associated with ***Topic*** because each quiz session is based on one selected topic. The topic defines the subject and content of the quiz, while the quiz session uses it to retrieve questions. However, the topic itself is independent and can be reused in different quiz sessions.

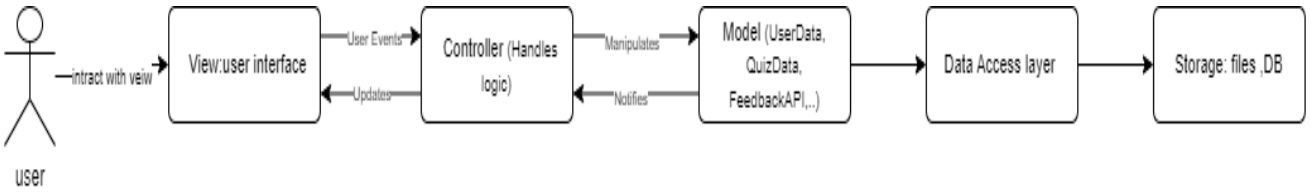
In addition, ***QuizSession*** is associated with ***ReportSummary*** because the quiz session generates and stores the result summary after completion. The report summary shows the outcome of a specific quiz session and is logically linked to it, but it is not a core component required for the quiz session to exist at all times.

Design Patterns & Principles

Design Patterns and Principles are essential in the Software Construction of an Educational Testing System because they provide structured, reliable, and reusable solutions to the system's recurring architectural challenges. In such systems where users interact with quizzes, submit answers, receive automated feedback, and store performance data the complexity grows quickly. Applying these design principles ensures the system remains scalable, maintainable, and easy to extend as new features such as adaptive testing, analytics, or additional user roles are introduced.

Using established Design Patterns minimizes code duplication, reduces tight coupling between components (such as the UI, quiz logic, and database operations), and supports the long-term sustainability of the system. These patterns also promote clearer system organization, making it easier for developers to update quiz logic, modify feedback algorithms, or change the user interface without disrupting other parts of the architecture. Overall, in the context of the Educational Testing System, Design Patterns and Principles contribute to building a robust, reliable, and future-ready learning platform.

Primary: Model View Controller (MVC) Pattern



The Model–View–Controller (MVC) pattern is applied in the Educational Testing System to ensure a structured, maintainable, and scalable architecture. The system separates the user interface, application logic, and data management into independent components, allowing them to evolve and operate effectively without creating unnecessary dependencies.

In this project, the interaction between system components follows the MVC sequence as described below:

1. **View (User Interface)**
   The View represents the graphical interface used by students and instructors.
   Examples include: login screens, quiz pages, result displays, and feedback forms.

Users interact directly with the View, such as selecting answers or submitting quizzes.

2. **Controller (Handles Application Logic)**

   When the user performs an action (e.g., starting a quiz, clicking “Submit”, requesting feedback),

   the event is sent to the Controller.

   The Controller: Processes the user's request , Validates data ,Calls the appropriate Model functions , Updates the View based on outcomes (e.g showing quiz results)

3. **Model (UserData, QuizData, FeedbackAPI, etc)**

   The Model contains the core system data and business rules.

   In this project, examples of Model components include:

   UserData (authentication, profiles) QuizData (questions, options, scoring rules) FeedbackAPI (automated feedback generation)

4. **Data Access Layer**

   The Model sends or retrieves information through a **Data Access Layer (DAL)**.

   The DAL is responsible for: Executing database queries , Ensuring secure storage Managing persistent data structures

   This layer isolates the database logic from the rest of the system, improving maintainability.

5. **Database**

   All final data such as quiz results, user records, and feedback history is stored in the database through the DAL. This ensures long-term persistence and allows future retrieval for analytics or progress reporting.

Benefits of Using MVC in the Educational Testing System

Clear separation of concerns improves organization and readability.

Scalability allows new quiz types or user roles to be added without major redesign.

Maintainability is enhanced because ui, logic, and data layers are modified independently.

Reusability of Models across different interfaces (e.g web, mobile).

Testability increases as each layer can be tested separately.

Supplementary: Factory Pattern

The Factory Pattern is a creational design pattern that centralizes and abstracts object creation. Instead of creating objects directly, a factory class or method determines which object type to produce based on specific conditions or input.

In Software Construction, the Factory Pattern helps minimize coupling between components by handling object creation in a single place. This makes the system easier to extend because new classes can be added without modifying existing code. It also supports consistency in how objects are constructed, leading to more organized and flexible architectures

SOLID Principle

The SOLID principles are five foundational guidelines for designing maintainable and scalable software systems. They promote good design practices and help developers construct software that is easy to update and expand.

In Software Construction, the SOLID principles guide the creation of modules and components that minimize dependencies, encourage reuse, and reduce the likelihood of introducing bugs when new features are added. They form the backbone of clean, organized, and extensible codebases.

S - Single Responsibility Principle (SRP)

A class should have only one reason to change.

Benefit: Reduces complexity, simplifies testing, and improves code clarity.

O - Open/Closed Principle (OCP)

Software should be open for extension but closed for modification.

Benefit: Supports adding new features without breaking existing functionality.

L - Liskov Substitution Principle (LSP)

Subtypes must be replaceable with their base types without altering correctness.
Benefit: Ensures reliable inheritance and predictable behavior.

I - Interface Segregation Principle (ISP)

Interfaces should be specific and not force clients to depend on unnecessary methods.
Benefit: Leads to cleaner, more modular interfaces and reduces unnecessary coupling.

D - Dependency Inversion Principle (DIP)

Depend on abstractions rather than concrete implementations.
Benefit: Enables flexibility, easier testing, and supports IoC and dependency frameworks.

Inversion of Control (IoC)

Inversion of Control (IoC) is a software design principle where the control of object creation and dependency management is transferred from the application to an external container or framework. Instead of classes instantiating their dependencies directly, these dependencies are supplied automatically.

In Software Construction, IoC improves modularity and supports clean architectural design. By moving responsibility for dependency management to a framework, developers avoid tight coupling and can more easily test components using mock or substitute dependencies. IoC aligns with the Dependency Inversion Principle and is commonly implemented through Dependency Injection (DI) or Service Locators.

Chapter 4: Construction & Unit Testing

Core Logic

The application is built on a 3-Tier Architecture (Presentation, Service, Data Access), ensuring a clean separation of concerns and high maintainability. The core business logic is encapsulated within the Service Layer which effectively decouples the JavaFX frontend from backend implementations.

Key Components and Logic:

Service Layers:

AuthService: Acts as the gatekeeper for the system. It handles user authentication by validating credentials against the StudentDatabase. It also manages the global application state via the Session singleton, ensuring that once a user logs in, their context (profile, history) is accessible system-wide.

QuizService: Maintains the entire quiz lifecycle. It serves as a facade, handling the creation of QuizSession objects, processing answer submissions, and triggering the final report generation. This centralization allows the frontend to interact with complex quiz logic through simple method calls like *startQuizSession()* and *submitAnswer()*.

Creational Patterns for Scalability:

QuestionFactory: This class implements the Factory Pattern to encapsulate the complexity of creating different question types (*MultipleChoiceQuestion*, *TrueOrFalseQuestion*). This makes the system extensible; adding a new question type in the future (e.g., "Fill in the Blank") would only require updating this factory, without breaking existing client code.

TopicBuilder: Utilizes the Builder Pattern to construct complex Topic objects. Since topics contain nested QuestionBank objects and metadata, the builder provides a fluent interface to assemble these components step-by-step, ensuring objects are always in a valid state upon creation.

Data Persistence (Repository Pattern):

The system uses In-Memory Singleton Repositories (*StudentDatabase*, *TopicDatabase*) to mimic a persistent database. This design choice ensures data consistency across different GUI screens while allowing for easy swapping to a real SQL database in future iterations.

Unit Testing

Unit testing is the process of testing small, individual unit of the system in isolation to verify the code's function or method's correctness. It is executed to check whether the code functions as intended based on its logic. Unit test promotes modular code, ensures test coverage and finally saves time by helping developers to identify defects in the early phases.

Test Strategy

In this project, unit tests will be covering two focus areas which are system framework and code coverage. As for framework, tools used are JUnit 5 to ensure flexible validation of the core logic classes while code coverage percentage is calculated by using the formula below

n = Number of lines of code executed
m = Total number of lines of code in the system's component

Code Coverage % = $(n \div m) \times 100$

Below are 3 test case examples to demonstrate the usage of JUnit 5 in unit testing.

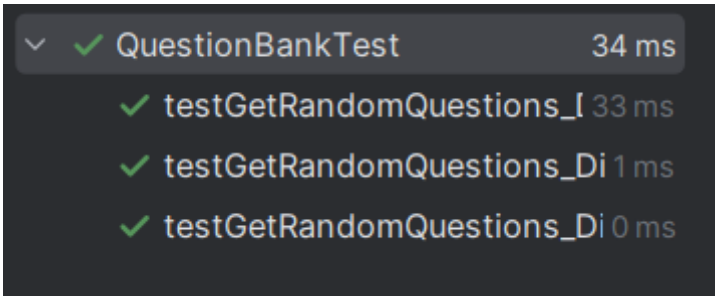
Question Bank

```java
@Test  Raphael Jean Irvin
public void testGetRandomQuestions_Difficulty1() {
    int numQuestions = 10;
    int difficultyLevel = 1;
    var questions = topic.getQuestionBank().getRandomQuestions(numQuestions, difficultyLevel);
    assert questions.size() == numQuestions;
    for (var question : questions) {
        System.out.println(question.getQuestionText() + " (Difficulty: " + question.getDifficultyLevel() + ")");
        assertTrue( condition: question.getDifficultyLevel() <= difficultyLevel);
    }
}

@Test  Raphael Jean Irvin
public void testGetRandomQuestions_Difficulty2() {
    int numQuestions = 10;
    int difficultyLevel = 2;
    var questions = topic.getQuestionBank().getRandomQuestions(numQuestions, difficultyLevel);
    assert questions.size() == numQuestions;
    for (var question : questions) {
        System.out.println(question.getQuestionText() + " (Difficulty: " + question.getDifficultyLevel() + ")");
        assertTrue( condition: question.getDifficultyLevel() <= difficultyLevel);
    }
}

@Test  Raphael Jean Irvin
public void testGetRandomQuestions_Difficulty3() {
    int numQuestions = 10;
    int difficultyLevel = 3;
    var questions = topic.getQuestionBank().getRandomQuestions(numQuestions, difficultyLevel);
    assert questions.size() == numQuestions;
    for (var question : questions) {
        System.out.println(question.getQuestionText() + " (Difficulty: " + question.getDifficultyLevel() + ")");
        assertTrue( condition: question.getDifficultyLevel() <= difficultyLevel);
    }
```

List of code



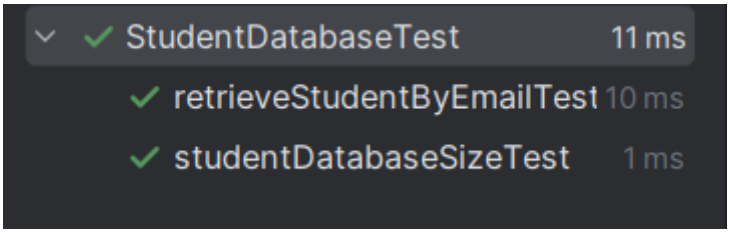
Result

This class verifies the logic for generating random quizzes based on difficulty. The line 'testGetRandomQuestion_DifficultyX' is a separation test between difficulty 1,2, and 3. The test request for 10 question on a specific difficulty level then loops through every question received to ensure that users actually got 10 questions and every question has difficulty level equals to requested.

Student Database

```
15
16      @Test  Raphael Jean Irvin
17      public void studentDatabaseSizeTest() {
18          int databaseSize = StudentDatabase.getInstance().getAllStudents().size();
19          assert (databaseSize == 3); // At least the three students created above should be in
20      }
21
22      @Test  Raphael Jean Irvin
23      public void retrieveStudentByEmailTest() {
24          Student retrievedStudent = StudentDatabase.getInstance().getStudentByEmail("a");
25          assert (retrievedStudent != null);
26          assert (retrievedStudent.getName().equals("a"));
27      }
28  }
```

Lines of code



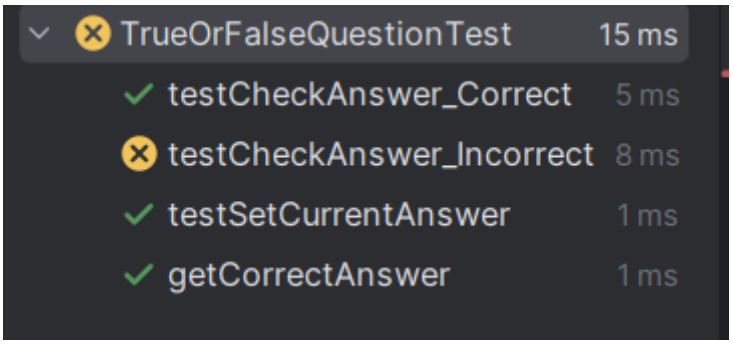
Result

This class test singleton database to ensure data persistence and retrieval work. The code 'studentDatabaseSizeTest' will retrieve all students and asserts the list size is 3. 'retrieveStudentByEmailTest' will search for a student via email and verify the result is not null and the name matches. Both test have the green checkmarks in the gutter, meaning that the database is intializing accurately with the expected dummy data.

True or False Question

```
21      @Test  Raphael Jean Irvin
22      public void getCorrectAnswer() { assertEquals( expected: "true", trueOrFalseQuestion.getCorrectAnswer()); }
25
26      @Test  Raphael Jean Irvin
27      public void testSetCurrentAnswer() { trueOrFalseQuestion.setCurrentAnswer("true"); }
30
31      @Test  Raphael Jean Irvin
32      public void testCheckAnswer_Correct() {
33          trueOrFalseQuestion.setCurrentAnswer("true");
34          assertTrue(trueOrFalseQuestion.checkAnswer());
35      }
36
37      @Test  Raphael Jean Irvin *
38      public void testCheckAnswer_Incorrect() {
39          trueOrFalseQuestion.setCurrentAnswer("true");
40          assertFalse(trueOrFalseQuestion.checkAnswer());
41      }
42  }
43
```

Lines of code

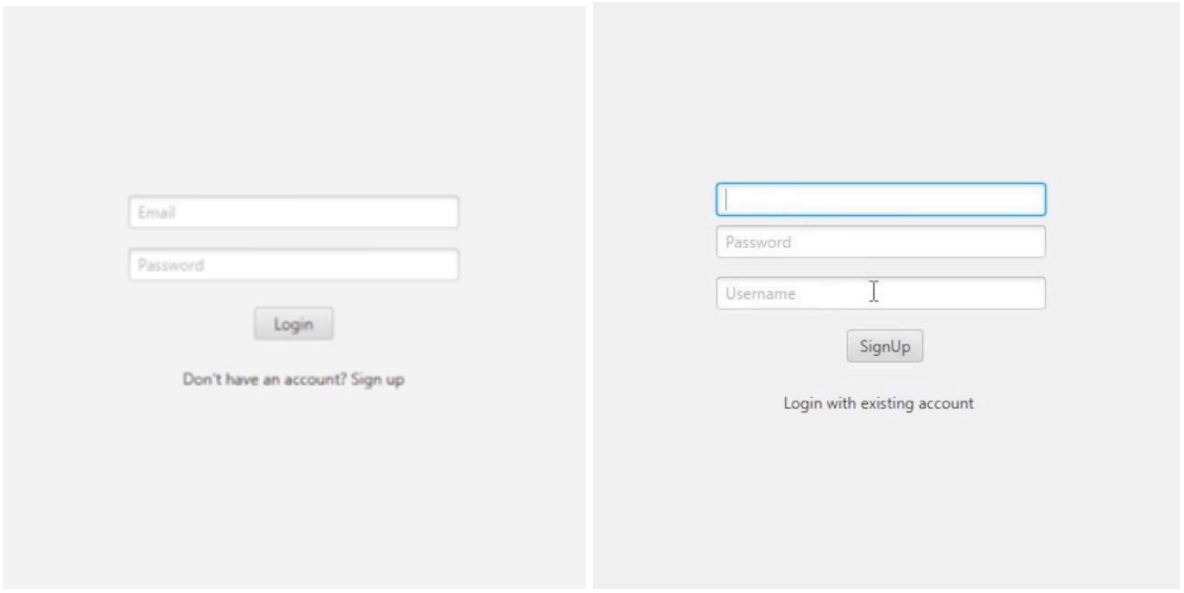


Result

This component test the logic of True/False question object. ‘testCheckAnswer_Correct’ will set the user’s answer to “true” and uses ‘assertTrue(..)’ to confirm that the system accepts its as correct. As for ‘testCheckAnswer_Incorrect’ will set the user’s answer to “false” and uses ‘assertFalse(...)’ to expect the system to reject it. In the reut, we can see that ‘testCheckAnswer_Correct’ passed with a green tick which means the logic holds up. As for ‘testCheckAnswer_Incorrect’, it failed with an orange cross because of the coding error. The answer should be set to “false” so that the expected ‘checkAnswer’ to be false. However, the result shows that it returned “true”. This shows assertion failure because the actual outcome did not match your expectation.

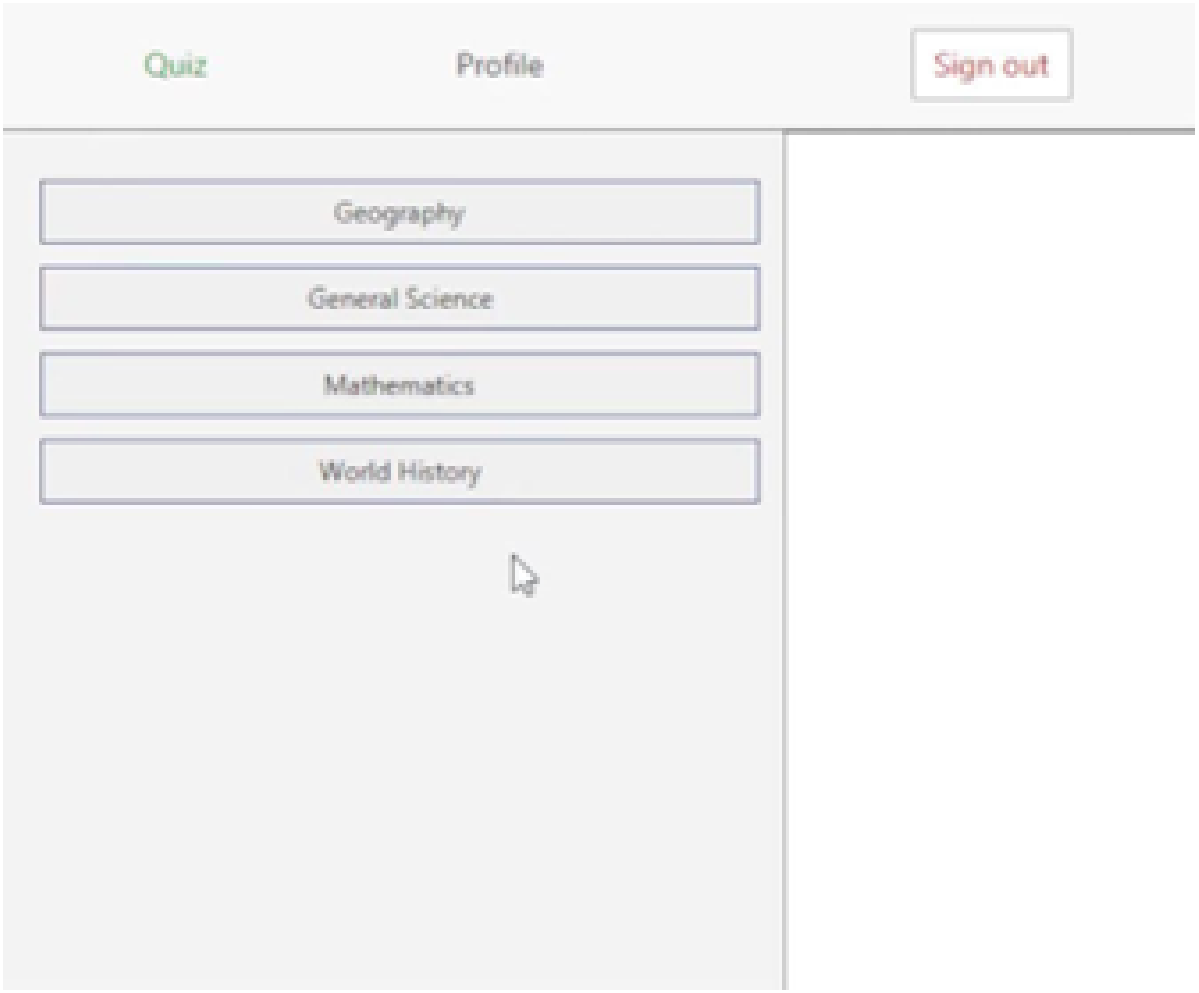
Chapter 5: Integration & Application

GUI Development



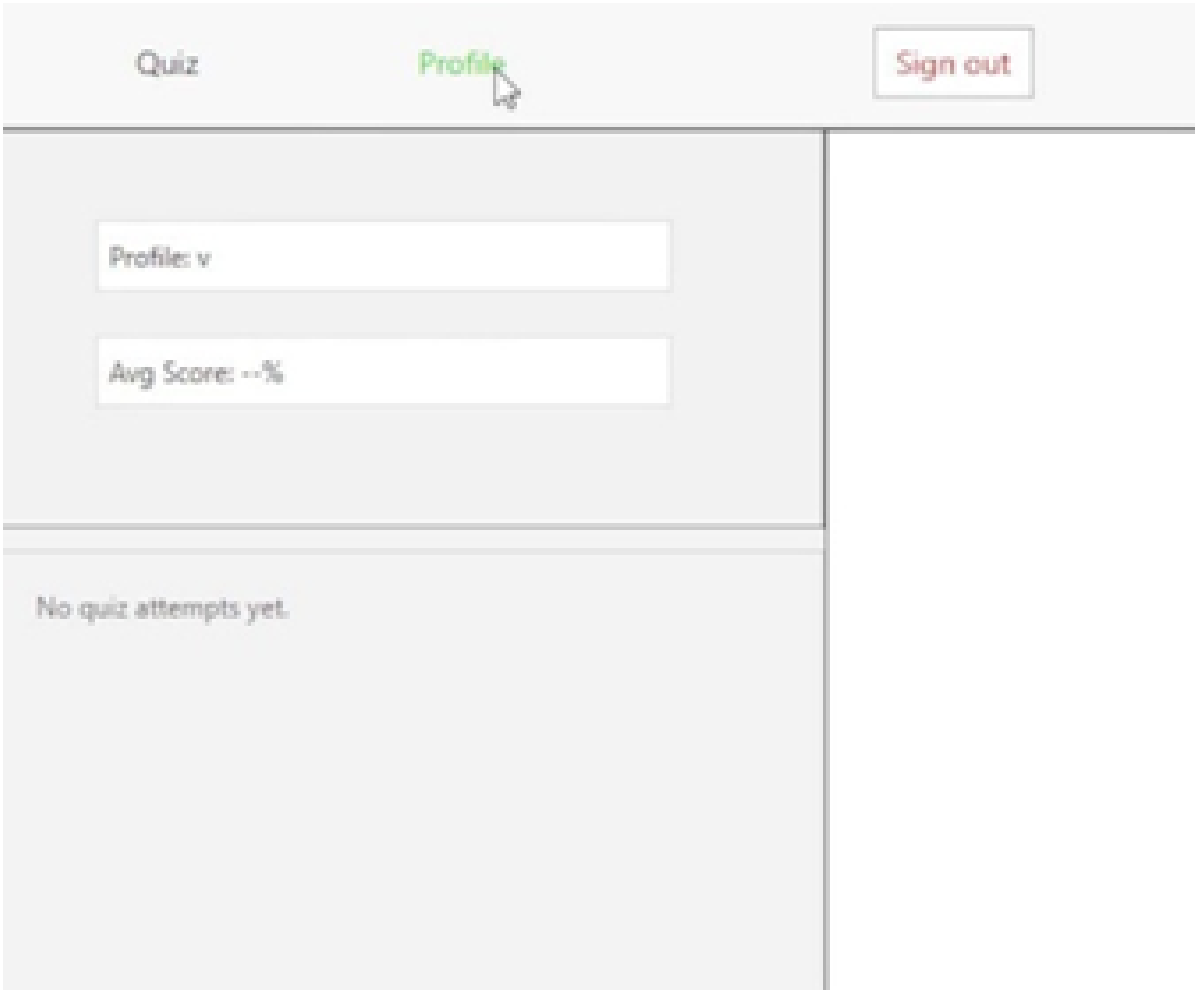
Authentication Interfaces

The authentication interfaces have a single and minimal design to make the login and registration screens consistent. The background is used in light colors and neutral color palette to maintain the cleanliness of the interface and avoid visual distraction. The fields of input are balanced visually with each other being centralized so that the necessary actions can be easily identified. There is also a placeholder text in the fields that gives instructions to the user on what should be input, with a very plain design with no additional labels. Primary actions (Login and Signup) are highly defined and placed below the input fields to have a natural flow of interaction. The secondary navigation items receive less visual emphasis to prevent a user being distracted by the secondary items. In general, the design provides transparency, uniformity, and usability, which are the purposes of an intuitive GUI.



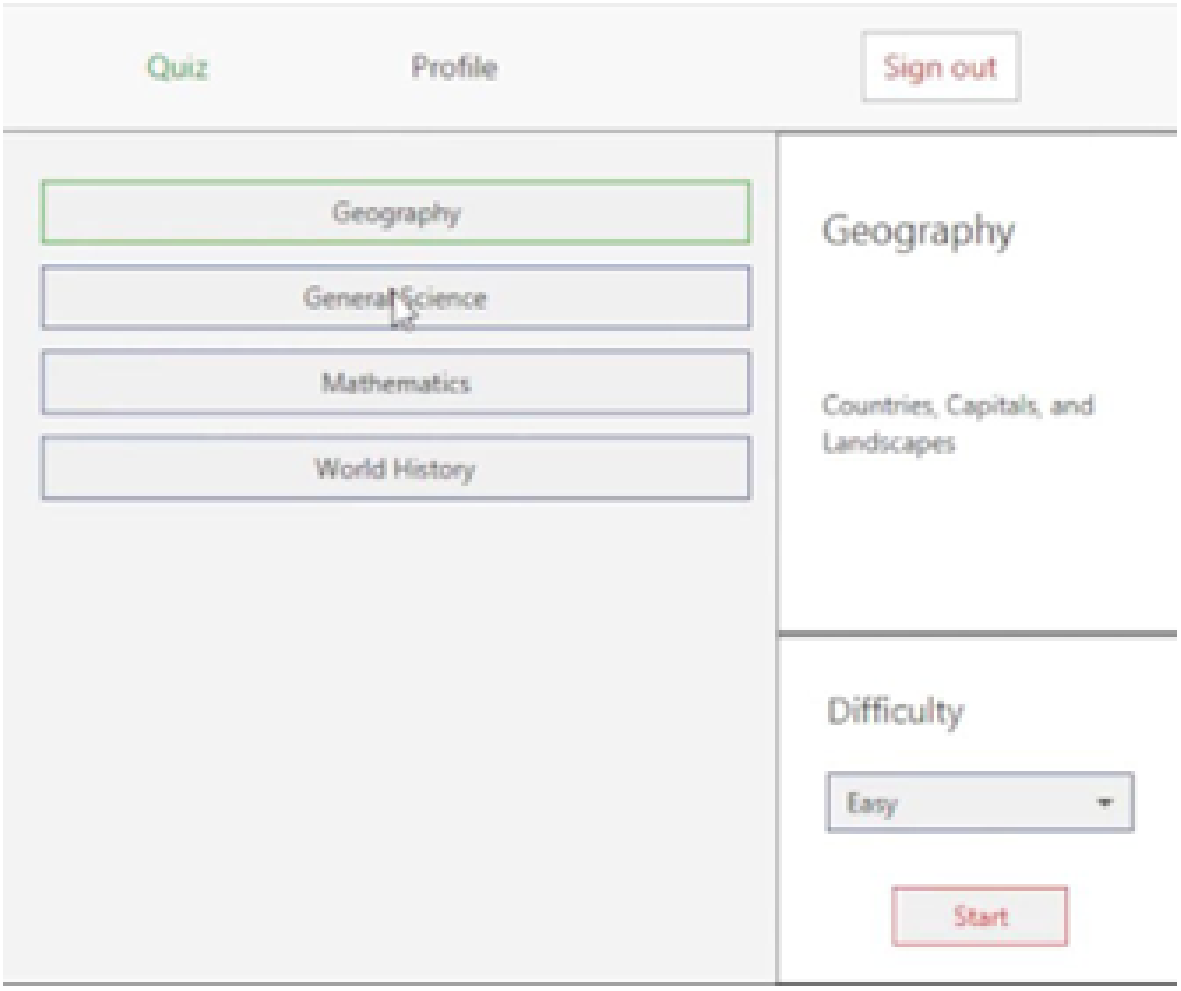
Home Interfaces

The home page should offer ease of access since direct access to the core of the system is offered. The top navigation bar is simple with only the separation of primary sections (Quiz, Profile) and the placement of the sign-out option in the visible and non-intrusive place. The categories of the quiz are shown in big and distinctly marked buttons to ensure easy and intuitive choice. The clear alignment by the vertical and the uniformity of the size of the button enhance easy reading and enable users to select a topic easily and with no confusion. The transparent colors and the uniformity of the lines are used to keep the appearance clean and focus on the elements of interaction. In general, the design is aimed at the clarity and ease of navigation, the rapid access to the main features, and intuitive user experience right after the login.



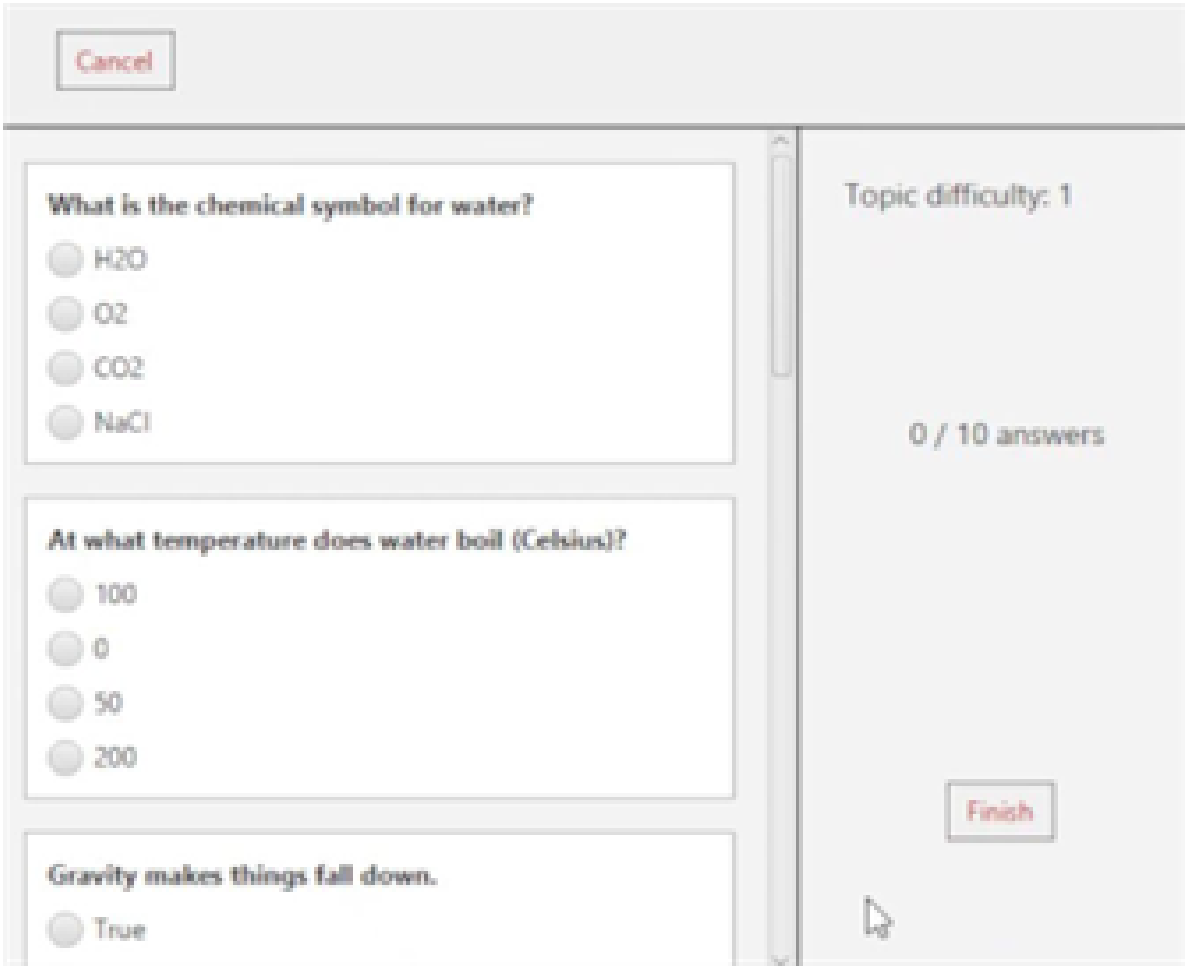
Profile Interfaces

The profile page is made in such a way that it display the user information in a nice and organized manner. It has a regular top navigation bar to enable easy navigation between the quiz and profile sections with the sign-out option always visible. The information about user and performance measures like average score are presented in distinct blocks to enhance legibility. The plain design and the lack of bright colors make the information rather than the design to be in the center of focus. That will have a quiz history section in order to track future progress, and a clear message in case no attempts can be made. This architecture will provide simplicity, consistency, and intuition whilst facilitating the objective of the system which is to observe the performance of the users with a clean and intuitive GUI.



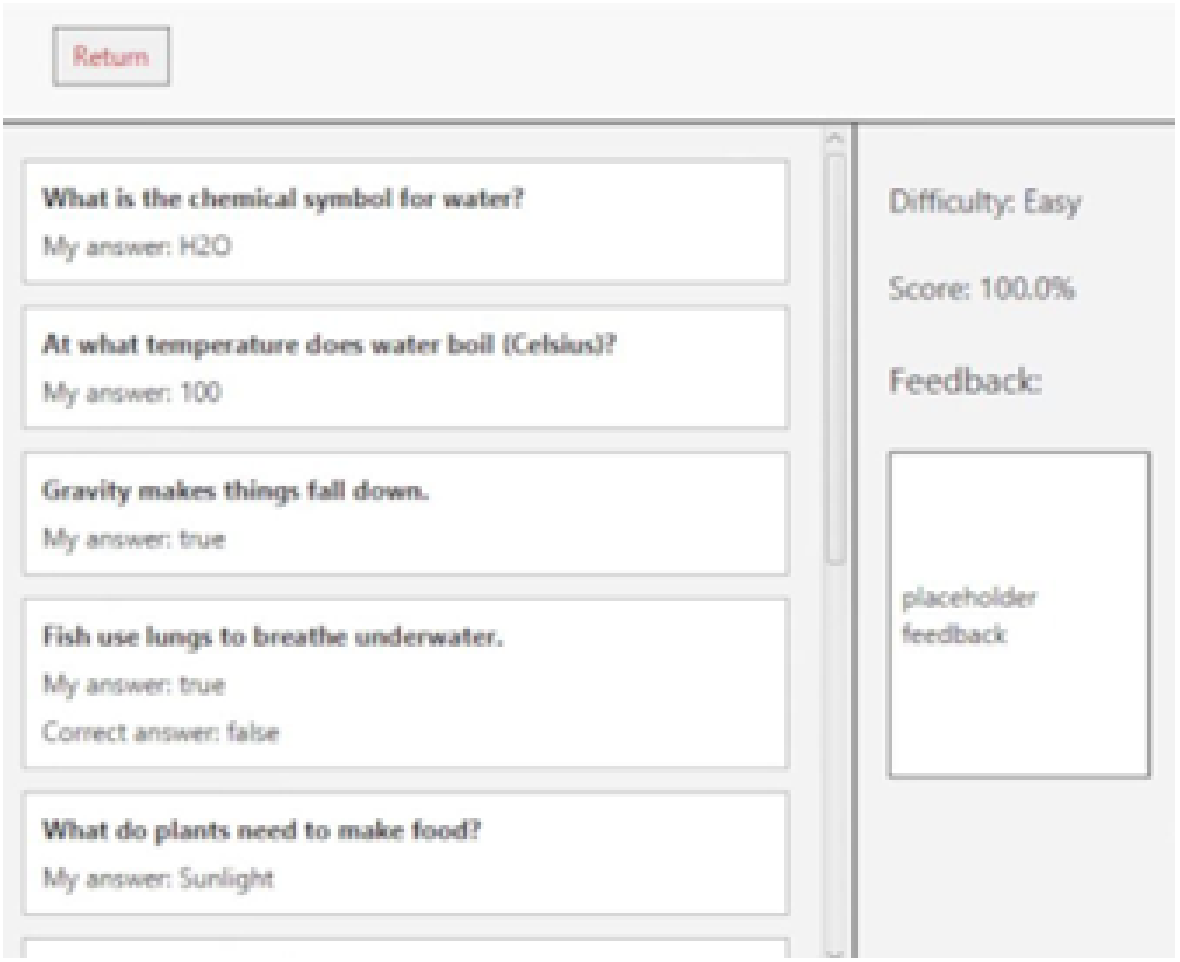
Quiz Selection Interfaces

The quiz page should be structured in such a way that it is easy to navigate the users through the process of choosing the quizzes. The categories of quiz appear in the form of huge, well-spaced buttons on the left to enable easy and fast navigation. The chosen category is visually placed to give instant feedback and ensure that the user chose the notification. The right side contains detailed information about the chosen quiz and helps to separate selection and configuration and enhances the readability. Difficulty dropdown is also provided to enable the user to have the ability to control quiz level, without overloading the interface. The bottom position of the start button is to adhere to a logical decision-making process of choosing a category to difficulty choice and action. This design helps to bring about clarity, user control and effective interaction such that the quiz process becomes intuitive and easy to follow.



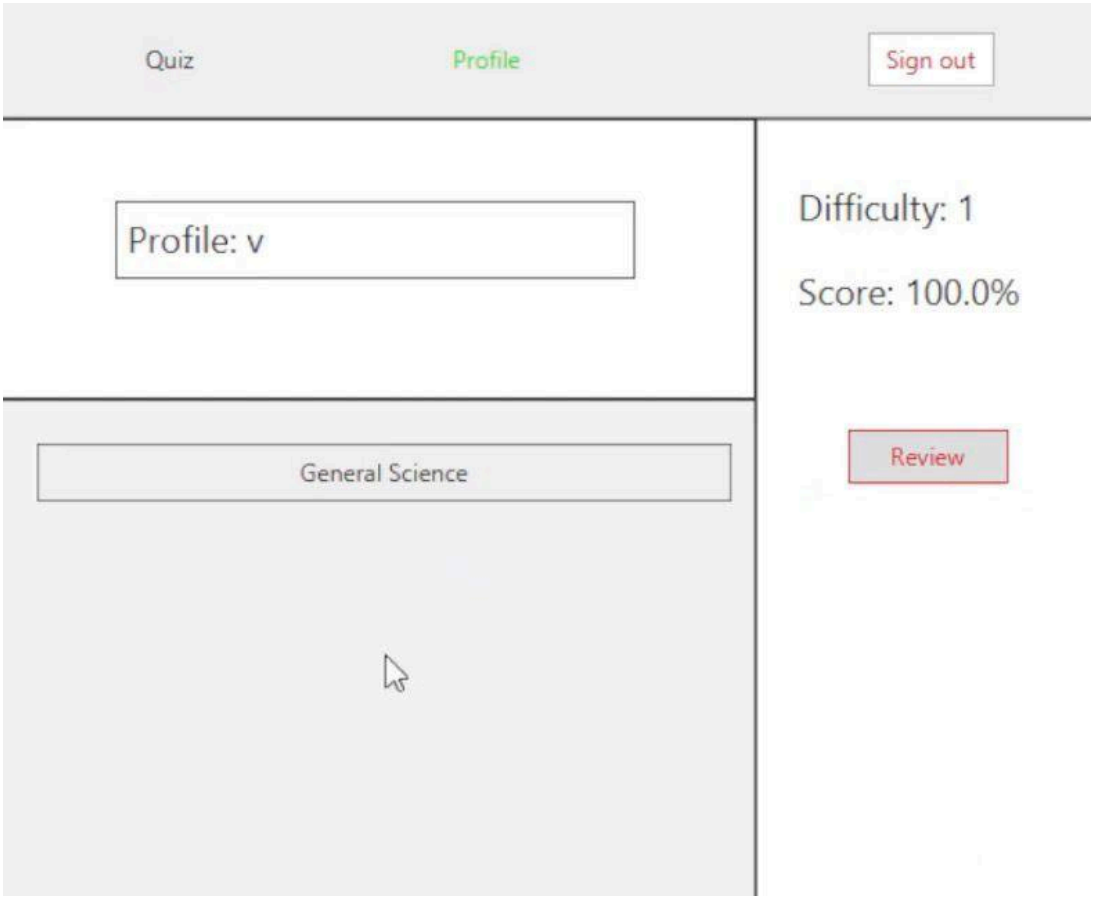
Quiz Question Interfaces

The quiz question page is set in such a way that the user does not get distracted in answering questions. The questions are shown in a clear vertical format and multiple-choice would be presented in the form of radio buttons because of which only one answer could be selected at a time. Multiple questions are addressed using a scrollable question box with a simple clean and organized interface. On the right hand side, there are progress indicators that indicate topic difficulty and the number of questions answered, which assists the user to monitor progress in the quiz. The control buttons like cancel and finish are distinctly separated to the question area to avoid unintentional clicks. In general, the design is focused on clarity, awareness of progress, and the simplicity of interaction, which helps to experience a seamless and intuitive process of taking the quiz.



Quiz Review Interfaces

The quiz review page will be used to provide an overview of the users performance after they have taken a quiz. Questions and answers provided by the user are being presented in a structured, scrollable list so that it is easy to review without any mess. The information on performance, including level of difficulty and final score, is displayed in a separate panel to have results in sight at all times. There is a clear differentiation between correct and incorrect answers which help in learning as well as self-assessment. There is also a feedback section where one can leave more comments or messages in the system. It has the return button at the top to provide a clear control to the users about navigation once the quiz is completed. In general, the design is focused on clarity, feedback, and learning support using a clean and organized GUI.



Quiz History Interfaces

The quiz history page will be created to show the previous quiz results in an easily readable format. User identity and quiz category are well displayed to allow context to be made of the presented results. Difficulty level and score are put in another panel to maintain a performance result and make it visible and comprehensible. There is a review button provided to enable the users to go back and reconsider the responses provided in the quizzes. The uniformity of the navigation bar makes it easy to move around the quiz and profile. In general, the design is aimed at the clarity and consistency and easy access to past performance in order to facilitate the effective tracking of the progress with the help of the clean GUI.

API Integration

To support SDG 4 (Quality Education), the system integrates an external Large Language Model (LLM) API to provide personalized, constructive feedback, transforming the assessment from a static score into a learning opportunity. In our application, the integration is handled by the *FeedbackGeneratorService* class, which connects to the OpenRouter API.

Request Construction: The system dynamically constructs a prompt based on the student's performance. Instead of sending generic requests, the buildPrompt method analyzes the *QuizSession* to identify specific incorrect answers. It compiles the question text, the student's wrong answer, and the correct answer into a structured context block. This allows the AI to generate targeted advice explaining why the student was wrong, rather than just stating the correct answer.

HTTP Client & JSON Parsing: The application uses Java's native java.net.http.HttpClient for network communication to minimize external dependencies. The JSON payload is constructed and parsed using the Google Gson library, ensuring robust handling of the complex nested JSON structures returned by the LLM API.

Error Handling: To ensure a smooth user experience, the API integration includes robust error handling mechanisms:

Status Code Validation: The service checks for HTTP 200 OK responses. If the API returns a 4xx or 5xx error (e.g., quota exceeded or server downtime), the system logs the specific error code.

Active Anticipation: A try-catch block wraps the entire network operation. In the event of a timeout, network failure, or parsing exception, the system catches the error and returns a fallback message (e.g., "Could not generate AI feedback at this time"). This prevents the entire application from crashing due to an external service failure, maintaining system stability.

System Testing

System testing is a comprehensive validation activity executed on a fully functional system to make sure that all requirements operate successfully as one system. The main goal of this process is to identify defects between the units that are integrated units and the whole system. The behaviour of a component is observed to validate its functionality.

Test Case ID	Input / Action	Expected Output	Actual Result	Status (Pass/Fail)
TC 01 - Login	User enters valid email and password, then click 'Login' button	User is redirected to main page	Successful login to main page	**Pass**
TC 02 - Login Failed	User enters valid email but invalid password	User will remain in the login page and it will display error text ('Invalid Credentials')	Successful display of error text and remains in login page	**Pass**
TC 03 – Sign up link	User clicks 'Sign Up' link on the login page	User is redirected to Sign Up page	Successful redirection to Sign Up page	**Pass**
TC 04– Sign up with existing data	User enters an existing email with other credentials and then clicks 'Sign Up' button	User will remain in Sign Up page and error text ('Username already exist') is displayed	Successful display of error text and remains in sign up page	**Pass**
TC 05 – Sign up	User enters valid email, password, and username, then clicks 'Sign Up' button	User is redirected to main page	Successful sign up to main page	**Pass**
TC 06 – Return link	User clicks on 'Return' link	User is redirected to Login page	Successful redirection to Login page	**Pass**
TC 07 – Topic selection	User clicks on 'Topic Title'	Topic attributes such as description, difficulty menu and 'Start' button is displayed	Successful display of topic attributes	**Pass**
TC 08 – Topic difficulty menu	User clicks 'Difficulty' dropdown menu to choose difficulty level (Easy/ Medium / Hard) and then click 'Start' button	User is redirected to Quiz page with chosen difficulty level	Successful redirection to Quiz page and chosen difficulty level	**Pass**

TC 09 – Level is not chosen	User ignore the 'Difficulty' dropdown menu and proceeds to click on 'Start' button	User will remain at Main page and error text ('Choose difficulty level') is displayed	Successful display of error text and user remain in the main page	**Pass**
TC 10 – Choose subject	User clicks on desired subject	The right pane of the Main page will display description of the subject, difficulty level menu and 'Start' button	Successful display of important details on the right side panel with	**Pass**
TC 11 – AI feedback	User completes all quiz question and view results page	The results page and personalized 'AI generated feedback' based on users answer	Custom feedback and AI-Generated improvements points are displayed	**Pass**
TC 12 – Submit quiz	User clicks on 'Finish Quiz' button	User is redirected to Score page where score and feedback is displayed	Successful redirection to Score page	**Pass**
TC 13 – Review Quiz	User completes a quiz and views the results page	Incorrect answers are highlighted with the correct answer displayed for review	Result page correctly reflects the completion status	**Pass**
TC 14 – History review	User clicks on a past quiz item in the profile history list	The application redirects the user back to detailed result page for that specific quiz.	Specific quiz result page is retrieved and displayed	**Pass**
TC 15 – Profile	User clicks on 'Profile' button	User is redirected to Profile page	Successful redirection to Profile page	**Pass**
TC 16 – Sign out	User clicks on 'Sign Out' button	User is redirected to Login page	Successful sign out to Login page	**Pass**

Chapter 6: Conclusion & Professional Practices

Demonstration Video

Demonstration Video Link

Reflection

How it supports SDG 4

The system of Educational Testing helps achieve SDG 4: Quality Education enhancing the manner in which students are evaluated and assisted in the learning process. The system also offers structured quizzes, instant results, and personalized feedback to show the students their weak areas and explain concepts better instead of being only based on final scores. The check of quizzes, monitoring of history, and the creation of feedback by artificial intelligence stimulate further learning and self-development. The system facilitates inclusive and effective learning by providing a convenient, accessible, and feedback-driven assessment experience to the various levels of learning.

Scalability potential

The system is created on an object-oriented basis and the Model-View-Controller (MVC) architecture and design patterns like Factory. This architecture enables the system to grow, but does not impact on the available components. The model or controller layers can be extended to add new forms of questions, new subjects, or use adaptive difficulty, or may also add external APIs. As the data access layer and core logic are decoupled, the system also extends to the databases or can be used in larger educational settings, like schools or universities, with little to no refactoring.

Teamwork roles, challenge, and solution

The project work was allocated to team members having definite roles, such as the development of the GUI, core logic implementation, testing, and documentation. One of the significant issues was the need to be consistent in the user interface and the application logic as well as combining various components. To solve this challenge, it is important to adhere to a modular design, have UML diagrams as a common point of reference and keep constant communication during integration. The team managed to effectively integrate personal input into a complete functional and unified structure through teamwork and definition of responsibilities

References

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall.

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Pearson.

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns.* O'Reilly Media.

Fowler, M. (2004). *Inversion of Control Containers and the Dependency Injection Pattern.* martinfowler.com.

Larman, C. (2004). *Applying UML and Patterns.* Prentice Hall.

Appendices

Appendix 1 - Source Code Github Repository

Github Repository

Appendix 2 - Demonstration Video

Demonstration (Final Video)

Appendix 2 - Unit Test Report

Module	Component	Result	Time	Coverage %
AuthServiceTest	registerTest_Success	Pass	0.05s	100%
	registerTest_Failure_DuplicateEmail	Pass	0.02s	100%
	loginTest_Success	Pass	0.04s	100%
	loginTest_Failure_WrongPassword	Pass	0.01s	100%
	loginTest_Failure_NonexistentUser	Pass	0.01s	100%
	sessionTest_AfterLogin	Pass	0.03s	90%
FeedbackGeneratorTest	testFeedbackGeneration	Pass	0.09s	75%
MCQTest	getCorrectAnswer	Pass	0.01s	100%
	testSetCurrentAnswer	Pass	0.01s	100%
	testCheckAnswer_Correct	Pass	0.01s	100%
	testCheckAnswer_Incorrect	Pass	0.01s	100%
QuestionBankTest	testGetRandomQuestions_Difficulty1	Pass	0.12s	92%
	testGetRandomQuestions_Difficulty2	Pass	0.11s	92%
	testGetRandomQuestions_Difficulty3	Pass	0.10s	92%
QuizFlowTest	mainTest	Pass	0.35s	50%
ReportFactoryTest	generateReport	Pass	0.15s	70%

StudentDatabaseTest	studentDatabaseSizeTest	Pass	0.08s	85%
	retrieveStudentByEmailTest	Pass	0.06s	95%
TopicDatabaseTest	testTopicStorage	Pass	0.05s	80%
TrueOrFalseTest	getCorrectAnswer	Pass	0.01s	100%
	testSetCurrentAnswer	Pass	0.01s	100%
	testCheckAnswer_Correct	Pass	0.01s	100%
	testCheckAnswer_Incorrect	Pass	0.01s	100%