

**UE22CS352B - Object Oriented Analysis & Design**

Mini Project Report on

[Quiz Management System](https://github.com/AbhishekShirol/Quiz-Hub) (QuizHub)

**Submitted by:**

**Name SRN**

**ABHIMANYU SINGH PES1UG23CS801**

**ABHISHEK SHIROL PES1UG23CS803**

**ABHISHEK POLICE PATIL PES1UG23CS802**

**ABHIJEET KUMAR PES1UG22CS009**

**Semester Section**

**6th A**

# Facultly Name

# Prof. Mahitha G

**January - May 2025**

##### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

##### PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

##### Problem Statement

##### [QuizHub](https://github.com/AbhishekShirol/Quiz-Hub) is a full-featured Quiz Management System designed for educational platforms. It supports role-based user interaction with auth, quiz creation, public ,private and filtered quizzes, real-time quiz attempts, and detailed quiz attempt history all in one seamless experience.

**Key Features:**

**🚀 Major Features**

**👥 Role-Based User Management with Auth**

* **Admin:**
  + **Manage all users**
  + **Create new admin accounts**
* **Educator:**
  + **Create quizzes (Private and Public)**
  + **View student analytics (quiz attempt history)**
  + **Add and manage questions with various difficulty levels**
  + **Set visibility and access options for quizzes as Public and Private**
* **Student:**
  + **Attempt quizzes using direct access or quiz codes**
  + **Track performance and review past attempts**
  + **Generate filtered quizzes based on preferences**

**📝 Quiz Creation**

* **Educators can create:**
  + **MCQs with multiple options and one correct answer**
  + **True/False Questions with binary response options**
* **Create Public and Private Quizzes:**
  + **Private quizzes require a quiz code to join**
  + **Public quizzes are accessible to all users in the question pool**
* **Add Hints to questions (using them reduces score)**
* **Configure quiz settings:**
  + **Set total time limit for Quiz**
  + **Customize quiz title and description**
  + **Define difficulty level for the entire quiz**

**🎯 Filtered Quiz System**

* **Students can generate custom quizzes by selecting:**
  + **Topic of interest**
  + **Difficulty Level appropriate to their knowledge**
  + **Number of Questions to include**
  + **Time for Quiz completion**
* **System workflow:**
  + **Based on the selected filters, the system searches the question database**
  + **If enough questions are available, creates the filtered quiz**
  + **If question shortage occurs, notifies student to adjust criteria**
  + **Allows students to save generated quizzes for later or start immediately**

**📊 Quiz Attempt Process**

* **Starting a Quiz:**
  + **Students can find quizzes through browsing or enter a quiz code**
  + **Quiz timer begins once the attempt is initiated**
  + **Questions are presented sequentially to the student**
* **During the Quiz:**
  + **Students read questions and consider answers**
  + **Option to use hints with score reduction penalty**
  + **Submit answers which are automatically saved**
  + **Progress through all questions until completion**
* **Quiz Completion:**
  + **Quiz ends when all questions are answered or timer expires**
  + **Automatic submission occurs if time limit is reached**
  + **System calculates final score based on correct answers and hint usage**
  + **Students can review results including correct answers**
  + **Performance data is stored for future progress tracking**

**🧩 Minor Features**

**🧠 Hint System**

* **Each question can include a hint.**
* **Students can use hints at the cost of reduced points.**

**🗃️ Question Bank Management**

* **Educators manage a question bank with:**
  + **Question & Options**
  + **Question type(MCQ & True or False)**
  + **Topic**
  + **Difficulty**
  + **Hints**

**⏲️ Timer Functionality**

* **Total quiz time can be enforced.**
* **Time taken on that Question**

**🔍 Search Functionality for PrivateQuizzes**

* **Users can search quizzes by:**
  + **The private Code Provided By the Educator**

**📈 Analytics Dashboard**

* **Educators can view detailed analytics of:**
  + **Quiz attempts for the quizzes**

## **Models Overview**

The database for the quiz application is designed using a relational schema. It consists of eight main tables, each serving a specific role in managing quizzes, questions, responses, and user data.

**1. Users**

* **Purpose:** Stores user credentials and roles.
* **Key Fields:**
  + **id:** Primary key, unique identifier for each user.
  + **created\_at / updated\_at:** Timestamps for record creation and updates.
  + **email & username:** Unique identifiers for user login.
  + **password:** Secured password storage.
  + **role:** Defines user type (ADMIN, EDUCATOR, STUDENT).

**2. Topics**

* **Purpose:** Defines the subjects or categories for quizzes.
* **Key Fields:**
  + **id:** Primary key.
  + **name:** Unique name of the topic.

**3. Quizzes**

* **Purpose:** Holds details about each quiz.
* **Key Fields:**
  + **id:** Primary key.
  + **title & description:** Basic quiz information.
  + **created\_at:** Timestamp of quiz creation.
  + **difficulty:** Overall difficulty level.
  + **time\_limit:** Duration allowed for the quiz.
  + **number\_of\_questions:** How many questions the quiz includes.
  + **filter\_criteria, private\_code, question\_ids, topics:** Additional configuration and organization details.
  + **visibility:** Determines whether the quiz is PUBLIC, PRIVATE, or FILTERED.
  + **user\_id:** Links to the educator or creator.

**4. Questions**

* **Purpose:** Contains individual quiz questions.
* **Key Fields:**
  + **id:** Primary key.
  + **question\_text:** The content of the question.
  + **difficulty:** Indicates question difficulty (EASY, MEDIUM, HARD).
  + **question\_type:** Specifies if the question is a multiple-choice (MCQ) or true/false.
  + **hint:** Optional hint for the question.
  + **created\_by:** References the creator (user).
  + **topic\_id:** Associates the question with a specific topic.

**5. Correct Options**

* **Purpose:** Stores the correct answer for each question.
* **Key Fields:**
  + **question\_id:** Links to the corresponding question.
  + **correct\_option:** The correct answer text.

**6. Question Options**

* **Purpose:** Lists all available answer options for a question.
* **Key Fields:**
  + **question\_id:** References the associated question.
  + **option\_text:** The text for each answer option.

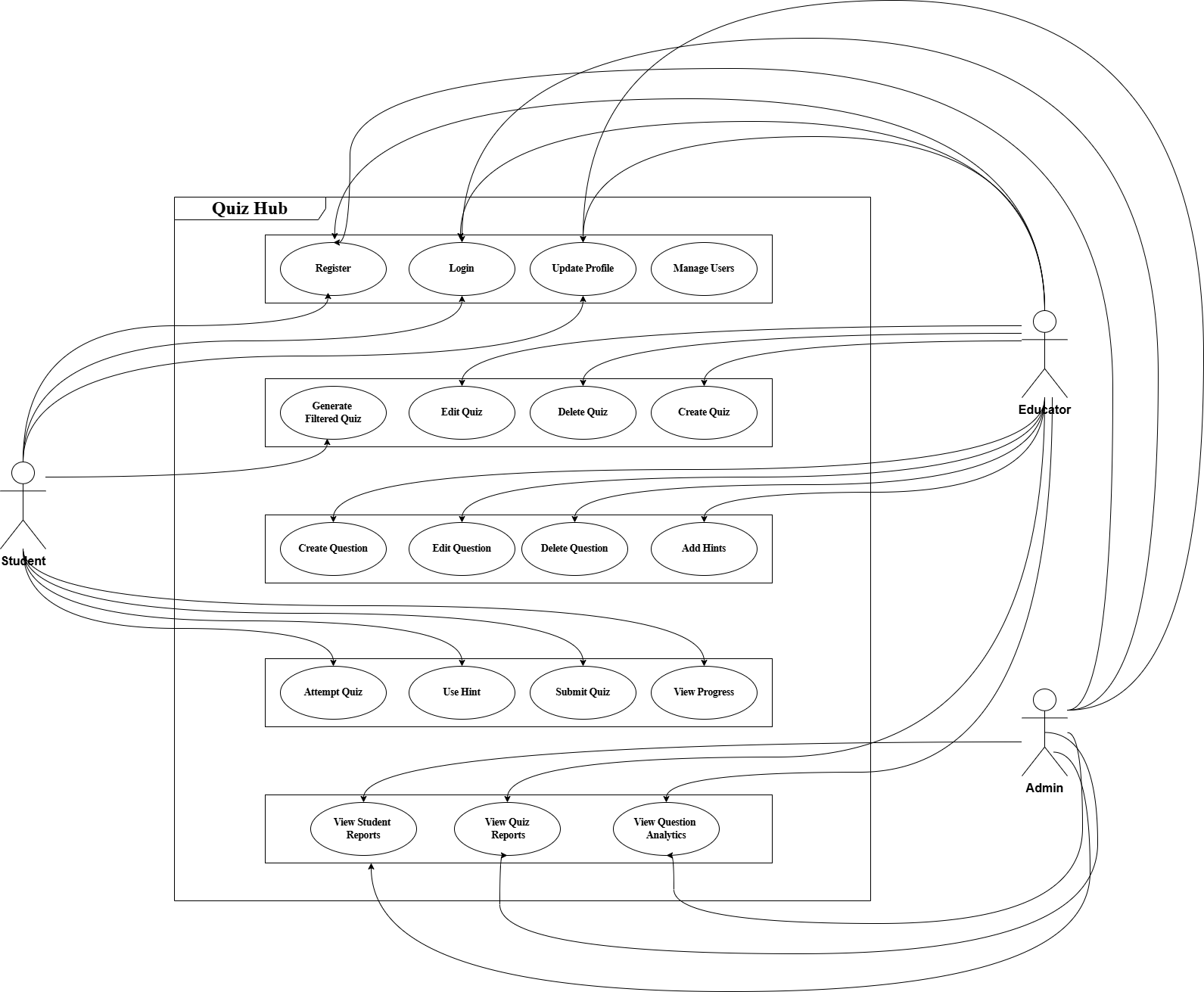
**7. Quiz Attempts**

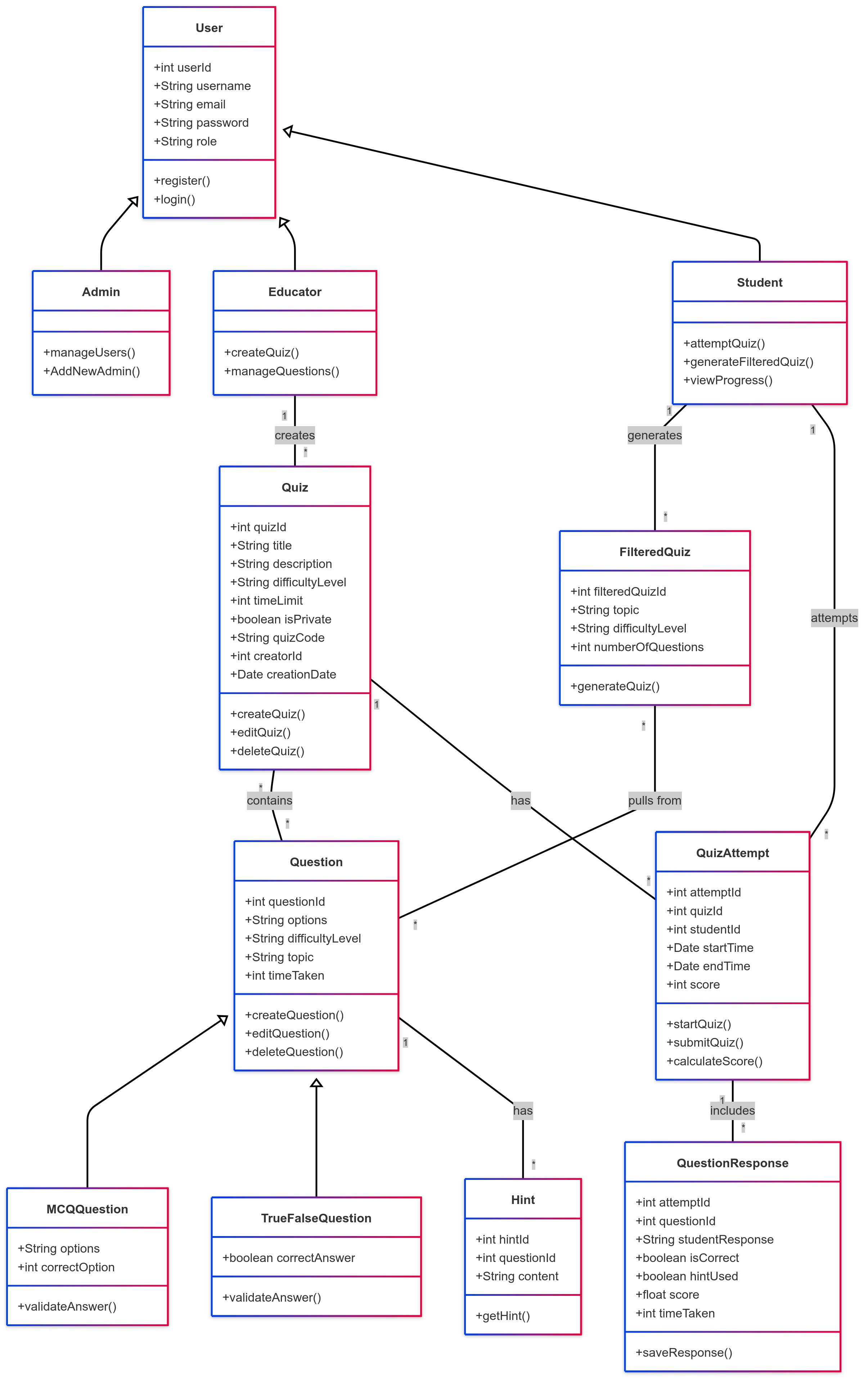
* **Purpose:** Tracks individual attempts of quizzes by students.
* **Key Fields:**
  + **attempt\_id:** Primary key for each quiz attempt.
  + **quiz\_id:** Identifies the quiz taken.
  + **student\_id:** Identifies the student taking the quiz.
  + **start\_time / end\_time:** Timestamps for the attempt duration.
  + **score:** Overall score achieved.

**8. Question Responses**

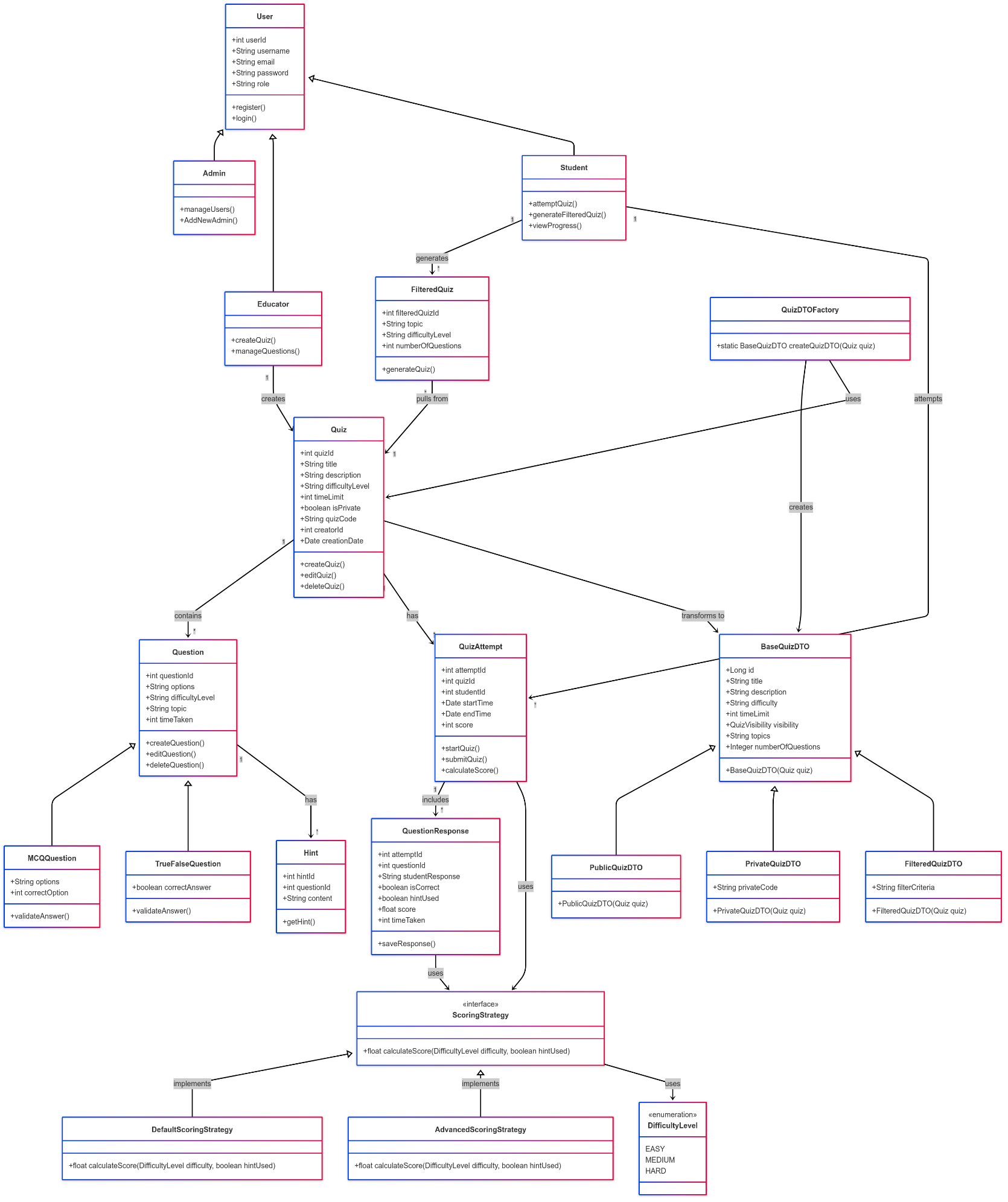
* **Purpose:** Records responses to individual questions within a quiz attempt.
* **Key Fields:**
  + **response\_id:** Primary key, uniquely identifies each response.
  + **question\_id:** Associates the response with a specific question.
  + **attempt\_id:** Connects the response to a particular quiz attempt.
  + **student\_response:** The answer provided by the student.
  + **is\_correct:** Indicates if the response was correct.
  + **hint\_used:** Flags whether a hint was used.
  + **score:** Score for the individual question.
  + **time\_taken:** Duration taken to answer.

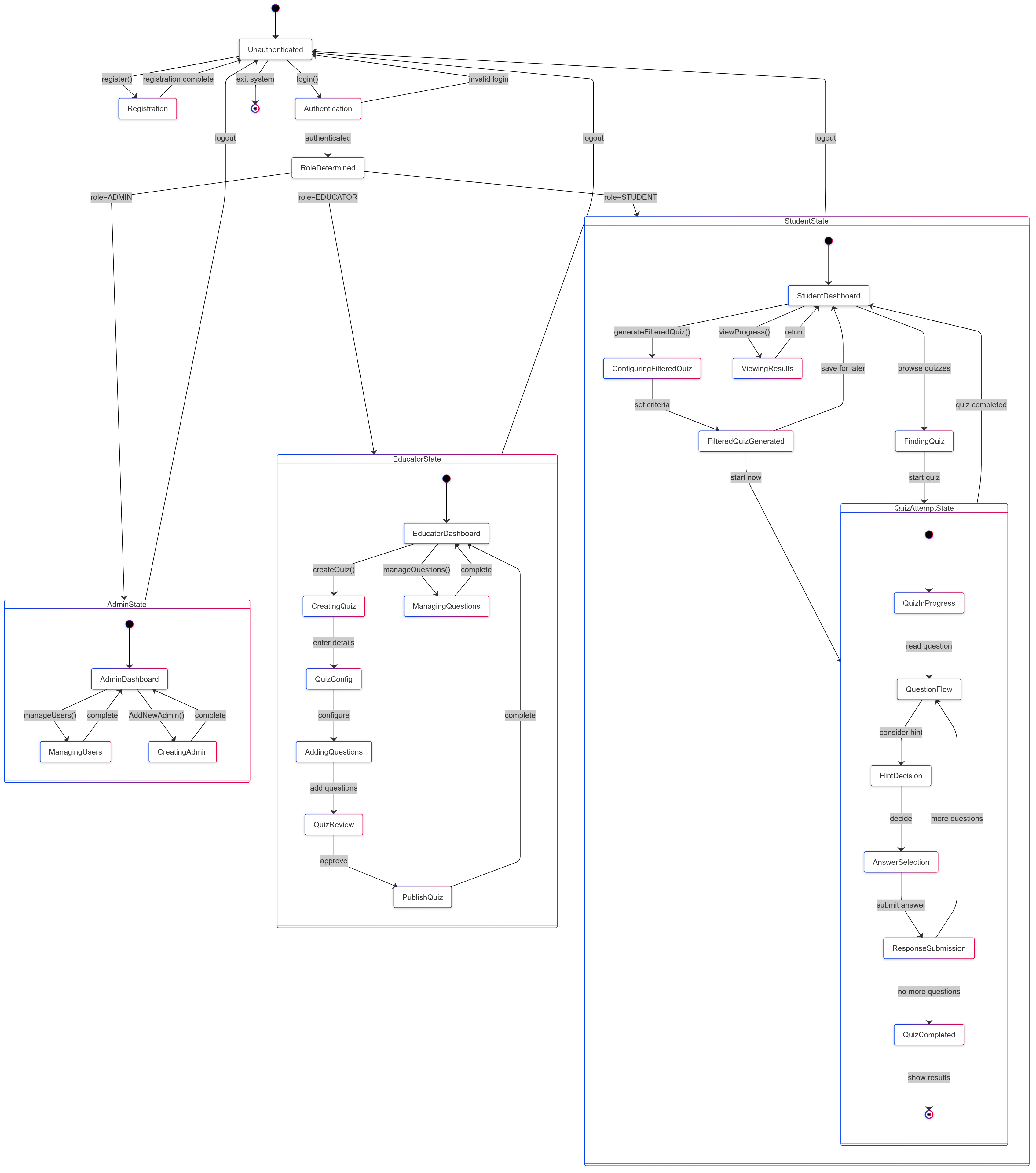
[**Use Case Diagram**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/UseCaseDiagram/UseCaseDiagram.png)

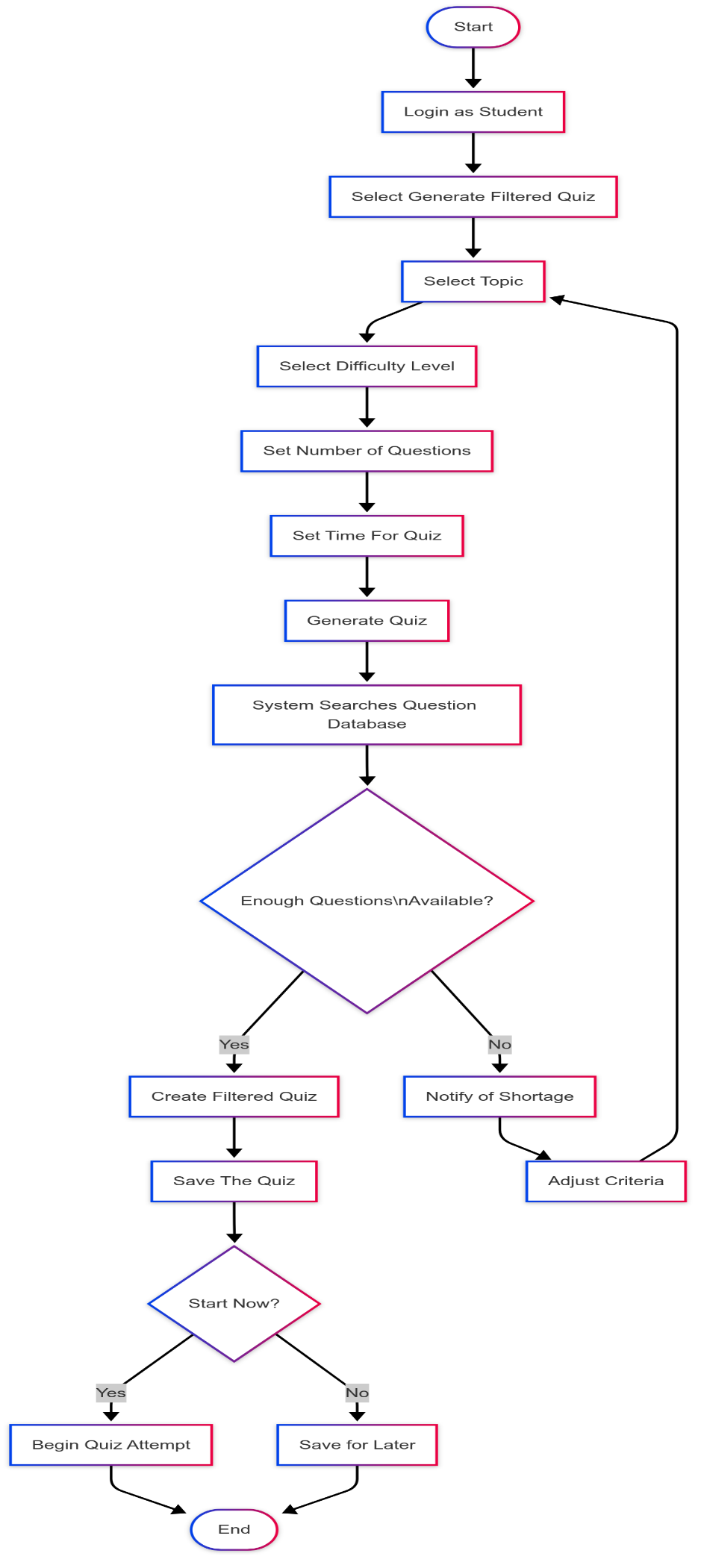
****

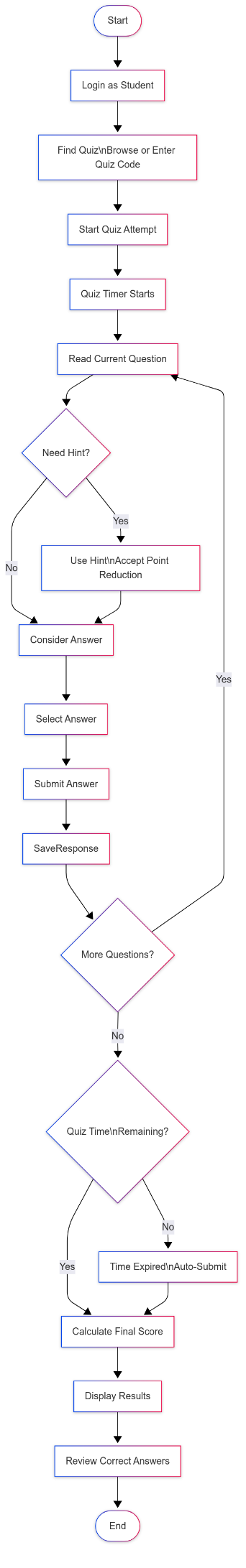
****[**Class Diagram**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ClassDiagram/ClassDiagram.png)

[**Class Diagram With Design Patterns :**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ClassDiagram/ClassDiagramWithDesignPatterns.png)

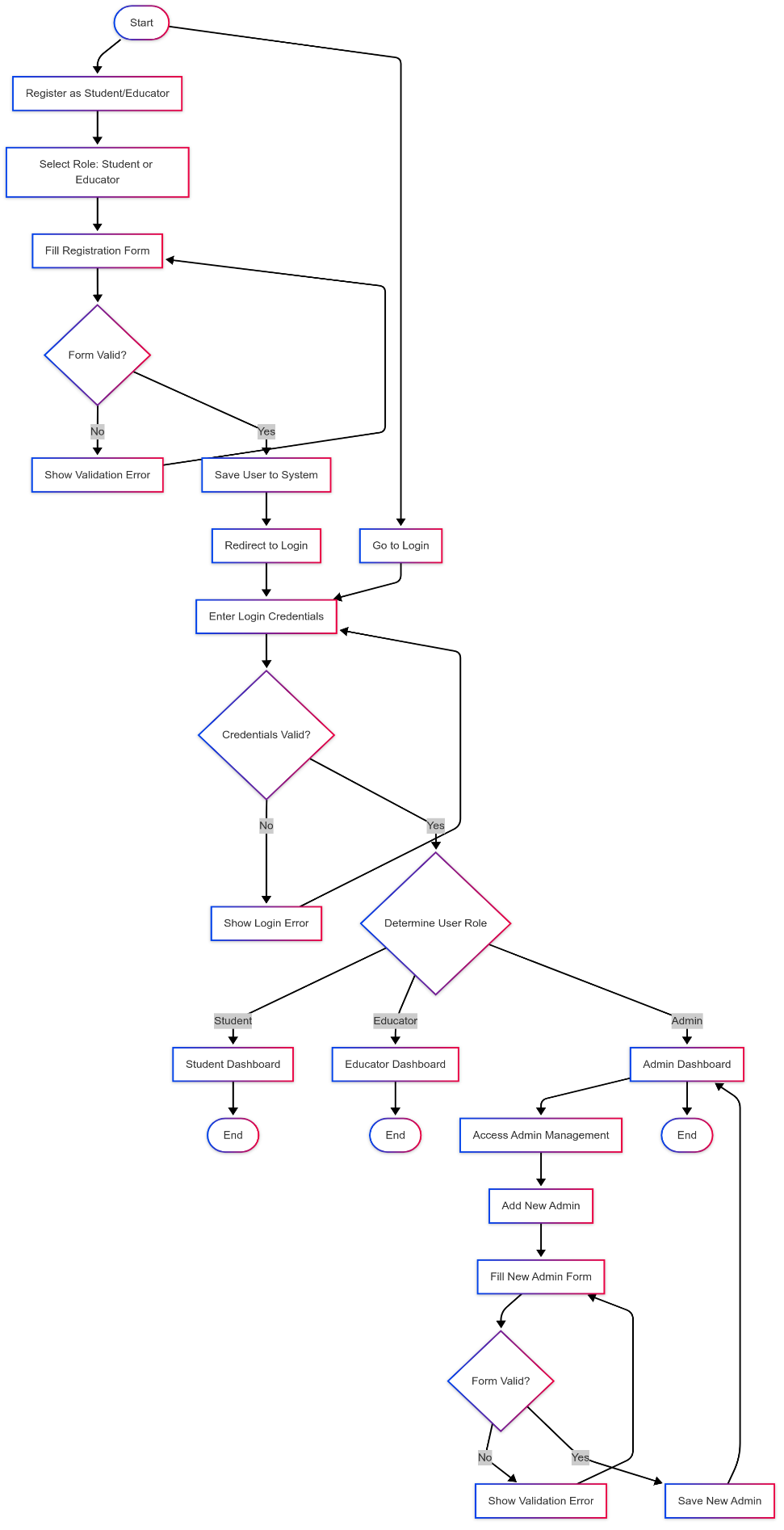
****

****[**State Diagram**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/StateDiagram/QuizHubStateDiagram.png)

[**Activity Diagrams**](https://github.com/AbhishekShirol/Quiz-Hub/tree/main/Diagrams/ActivityDiagrams)

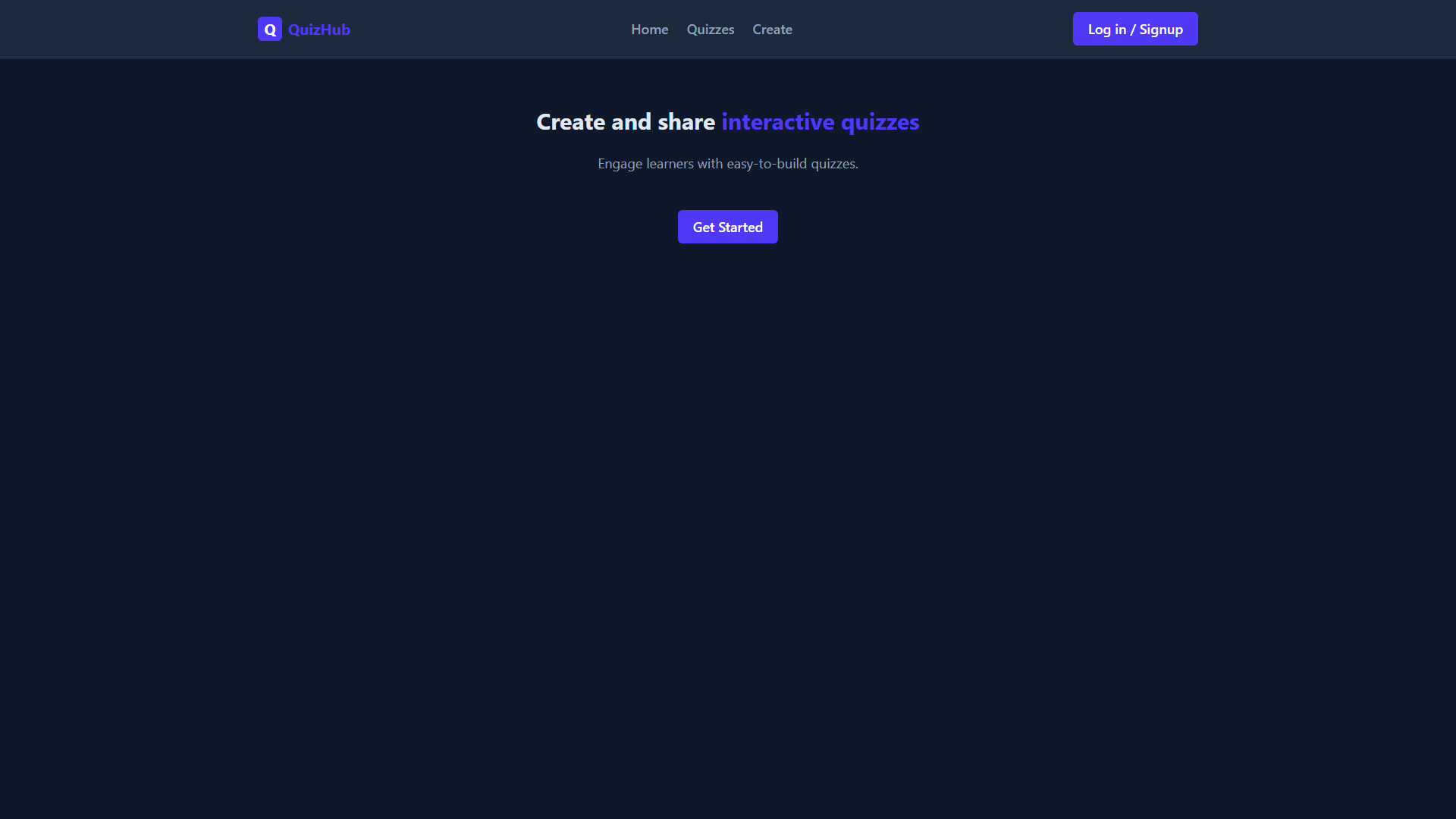
1. [**Filtered Quiz Creation**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ActivityDiagrams/FilteredQuizCreation.png)
2. **Qu**[**izAttempt**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ActivityDiagrams/QuizAttemptAD.png)
3. ****[**QuizCreation**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ActivityDiagrams/QuizCreationAD.png)

****

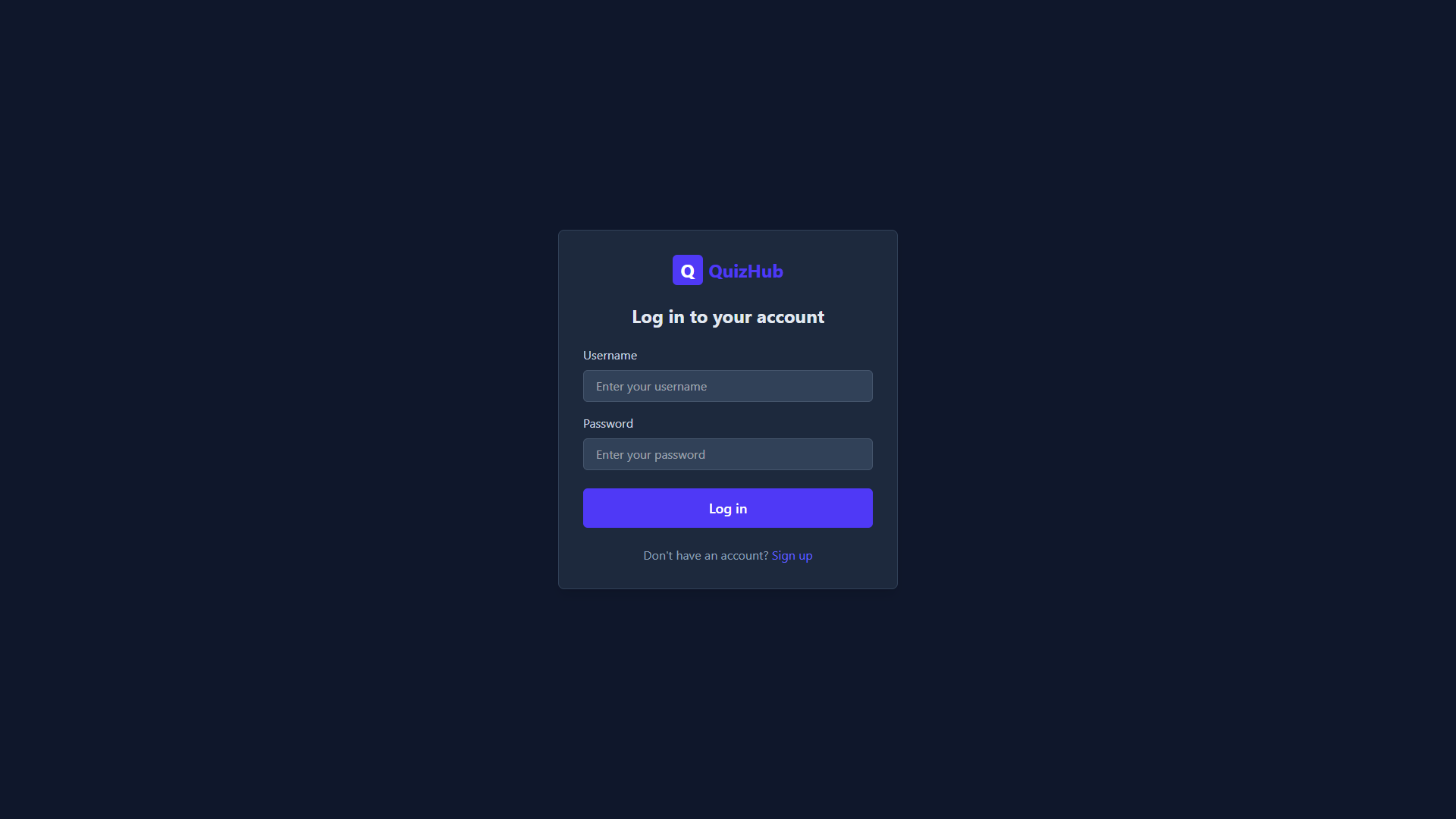
1. ****[**UserRegistrationAndLoginAD**](https://github.com/AbhishekShirol/Quiz-Hub/blob/main/Diagrams/ActivityDiagrams/UserRegistrationAndLoginAD.png)

**UI :**

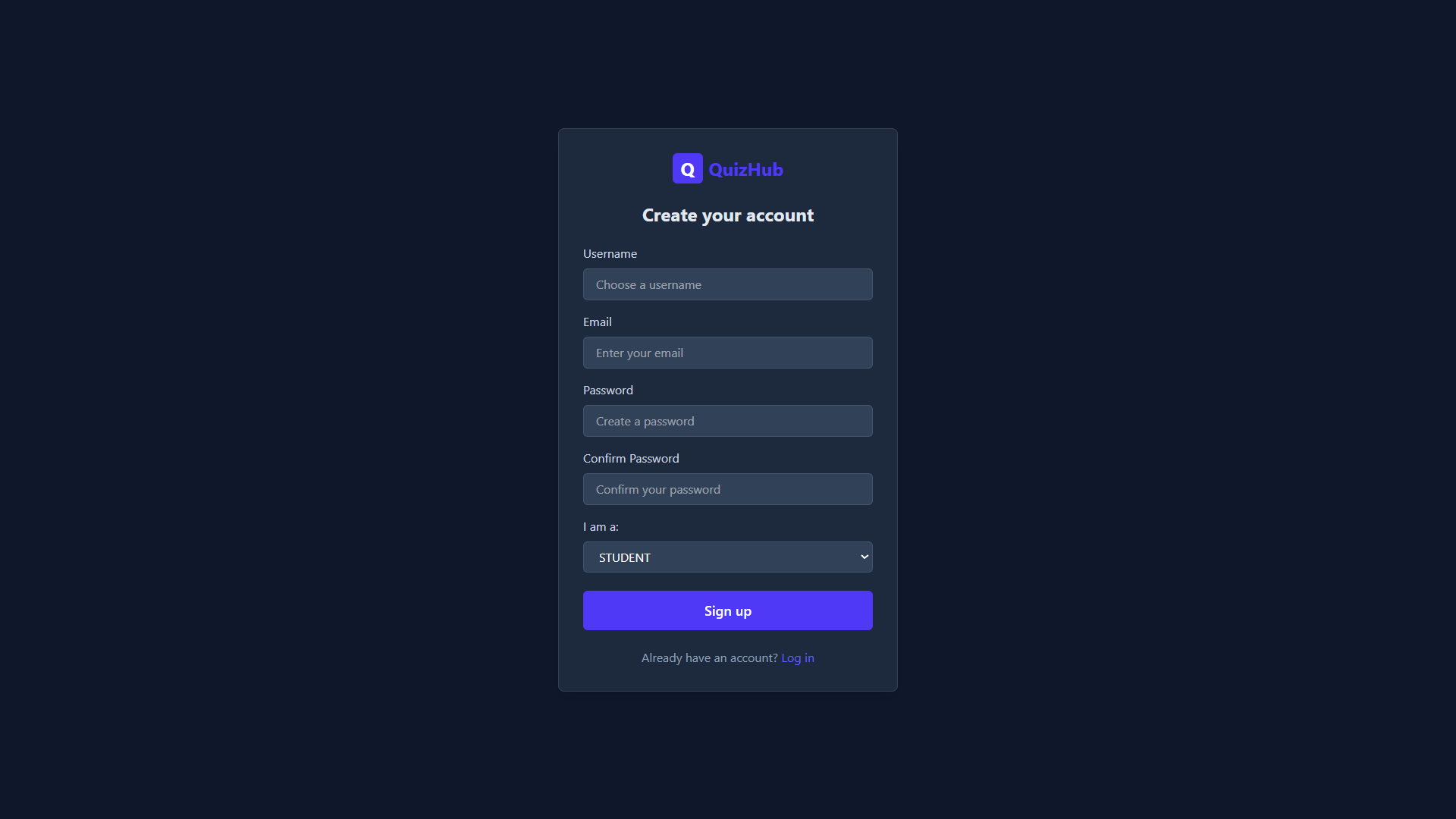
**Home page:**

****

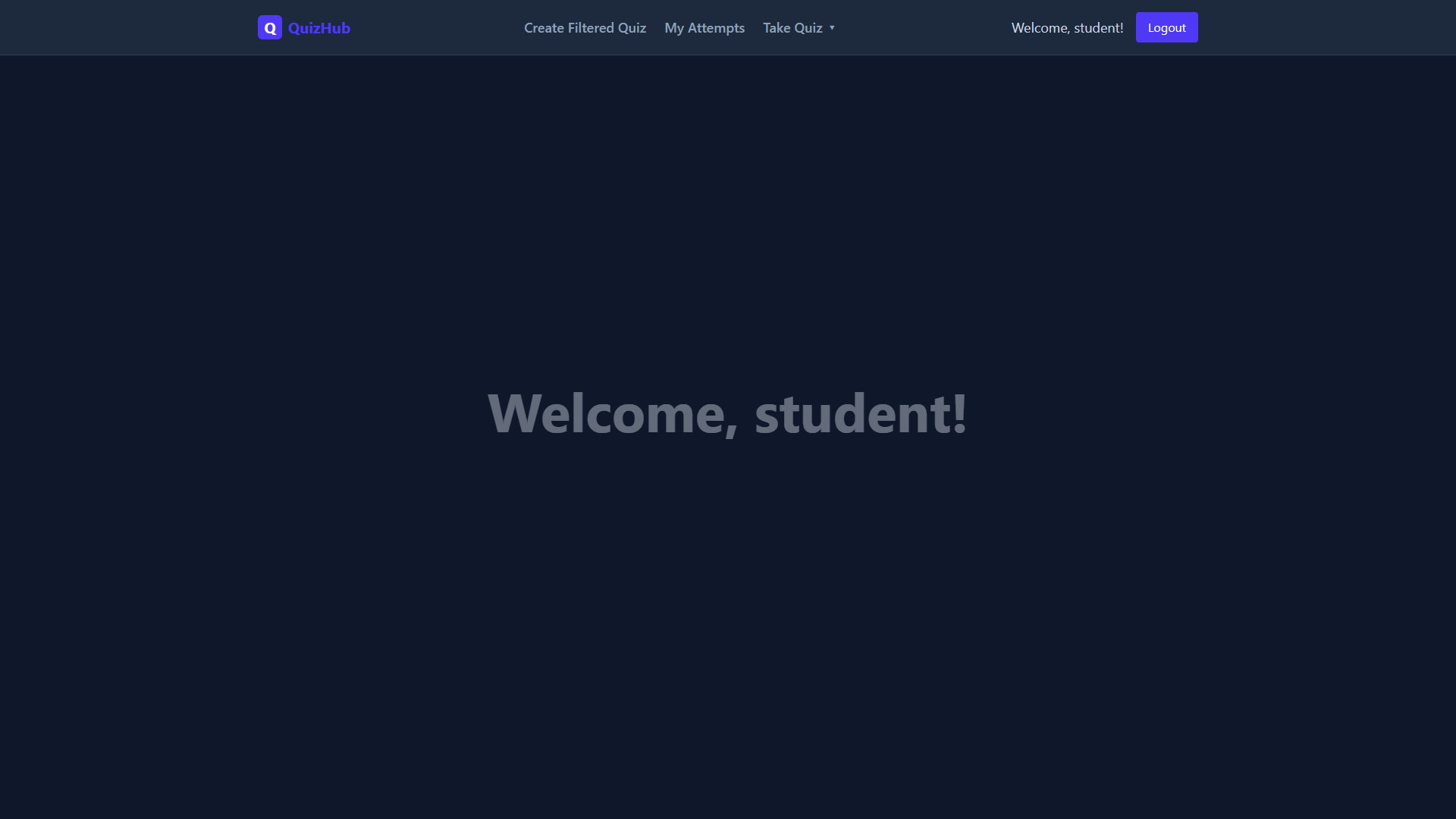
**Login:**

****

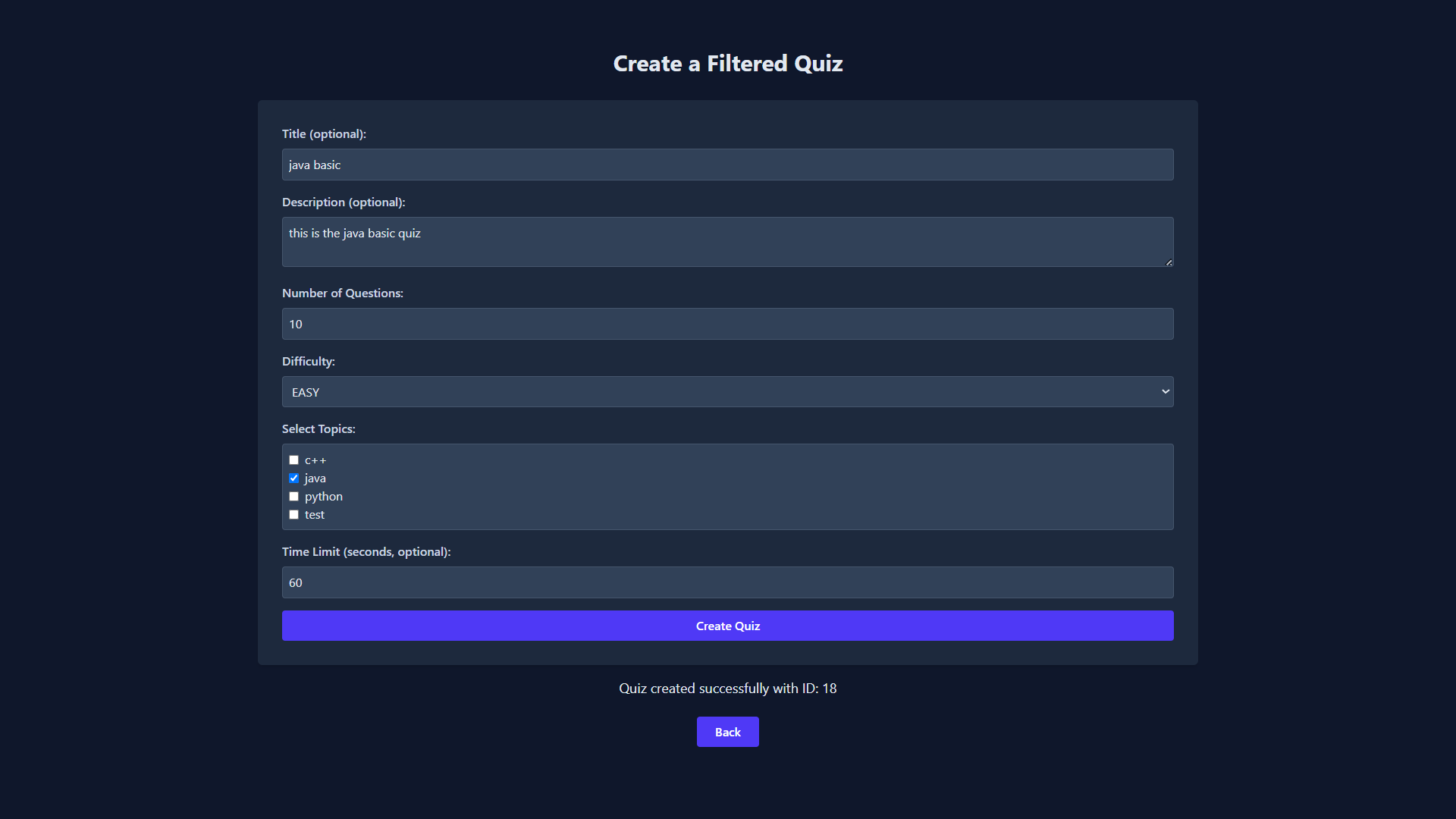
**Sign up :**

****

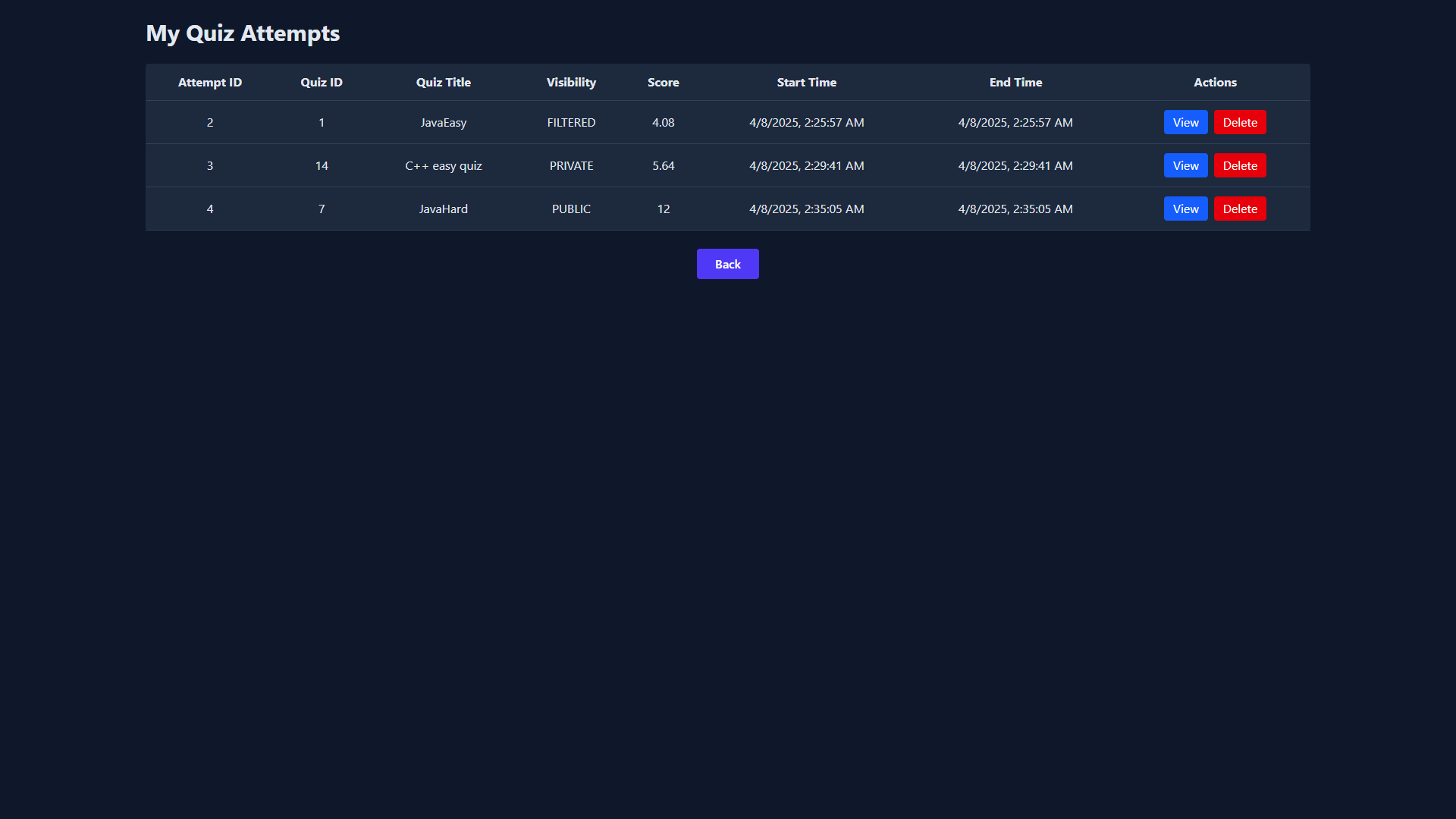
**Student Dashboard :**

****

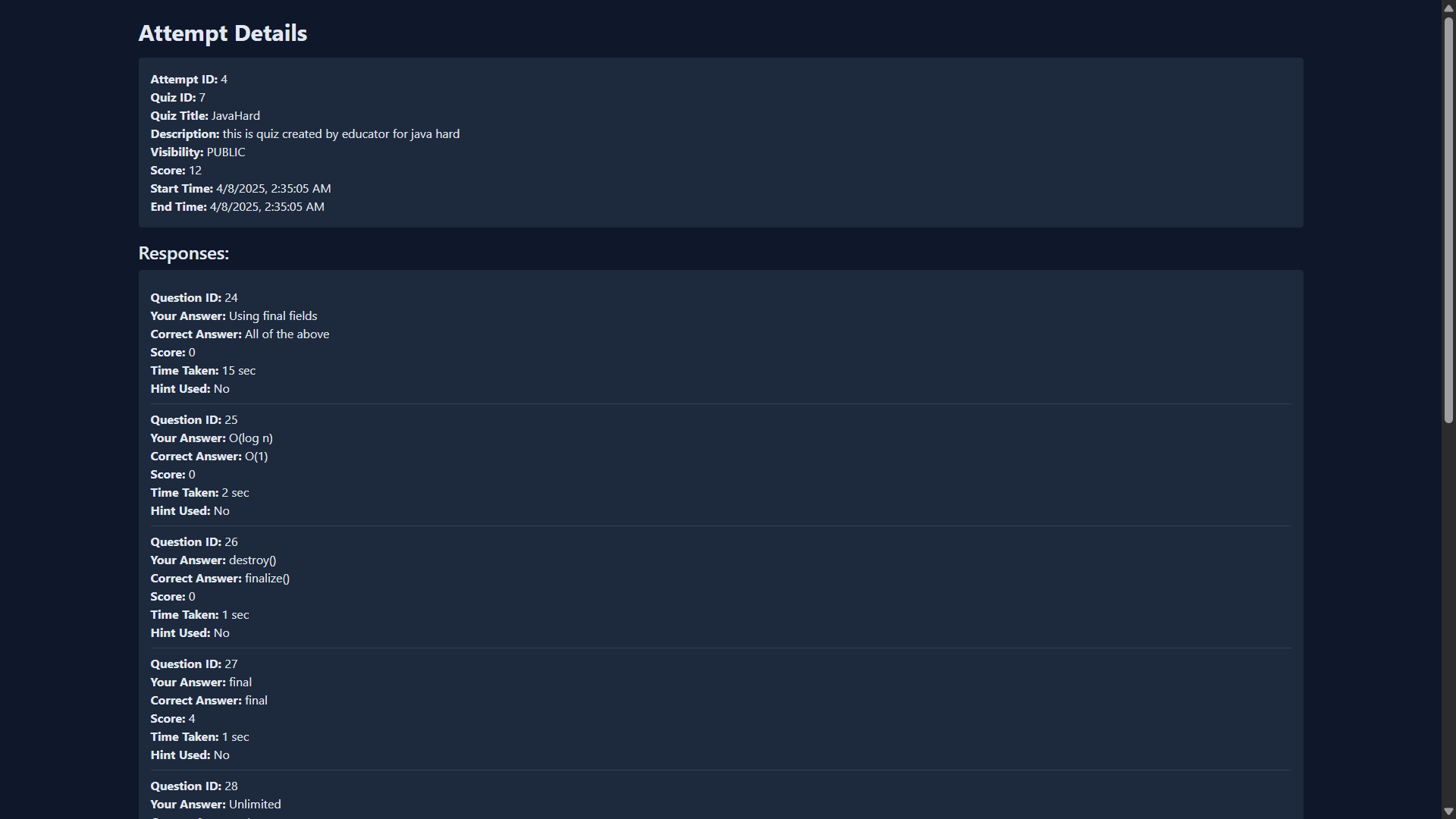
**Filtered Quiz Creation:**

****

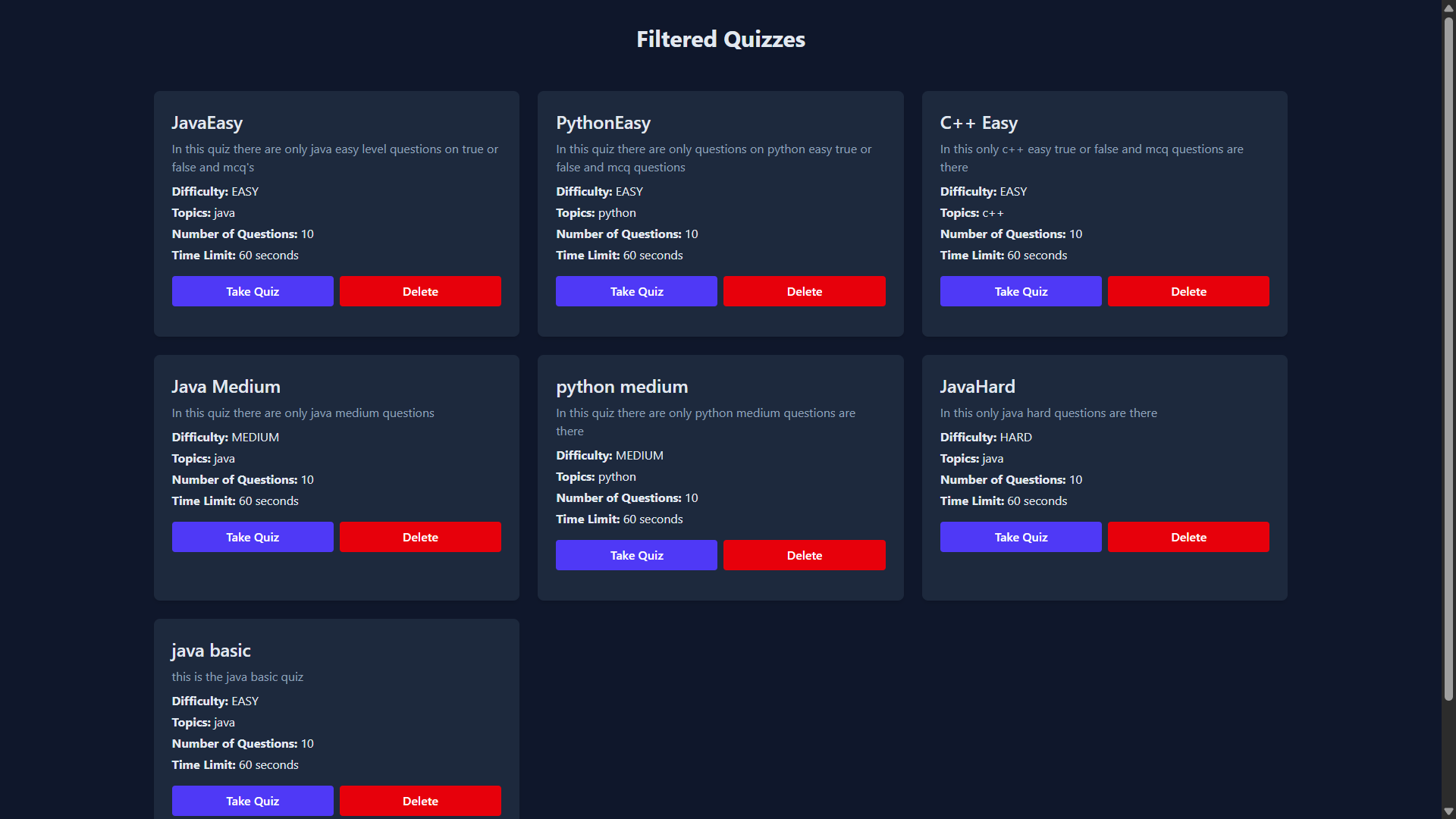
**Quiz Attempts:**

****

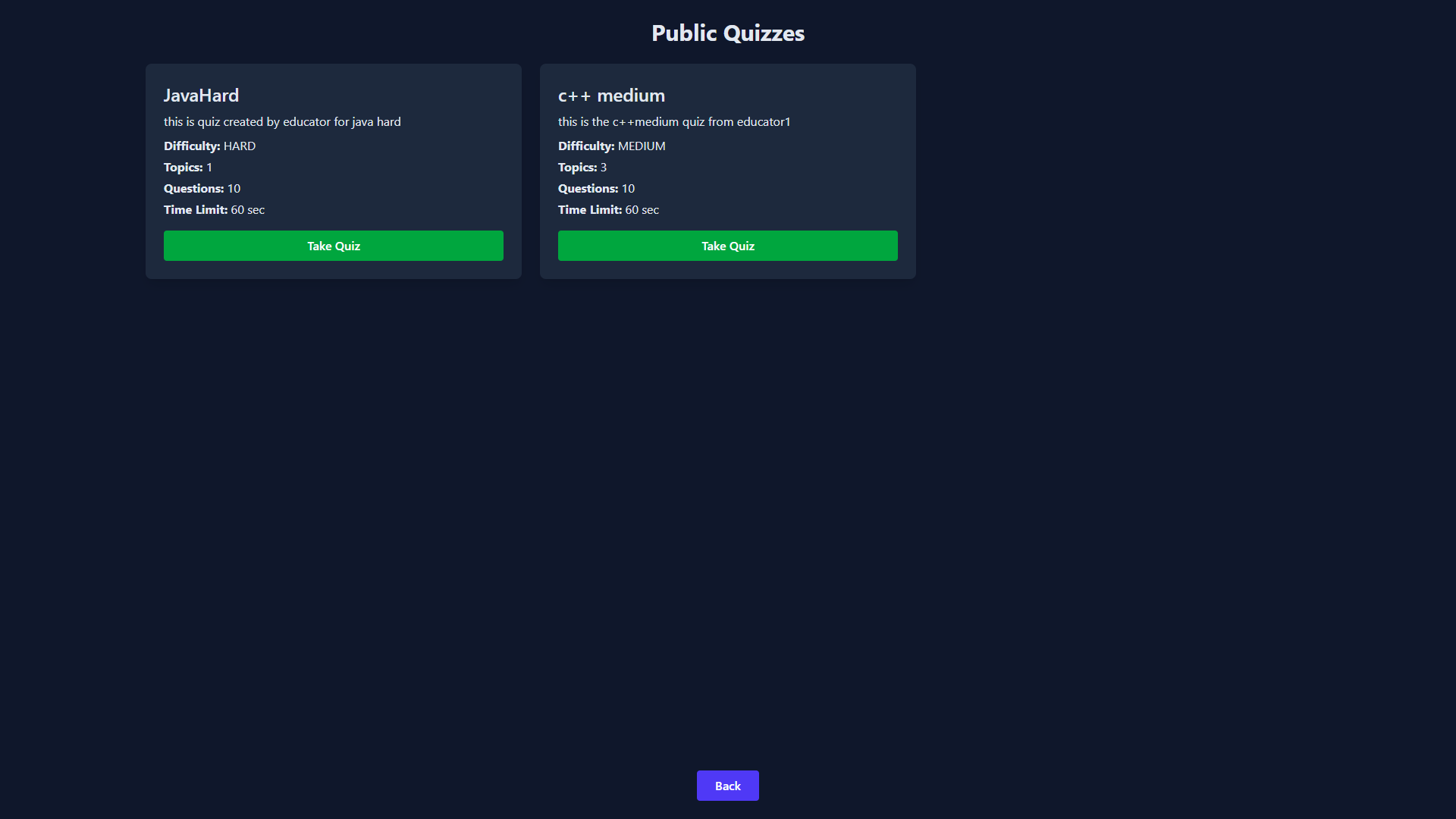
**Quiz Attempt Detailes :**

****

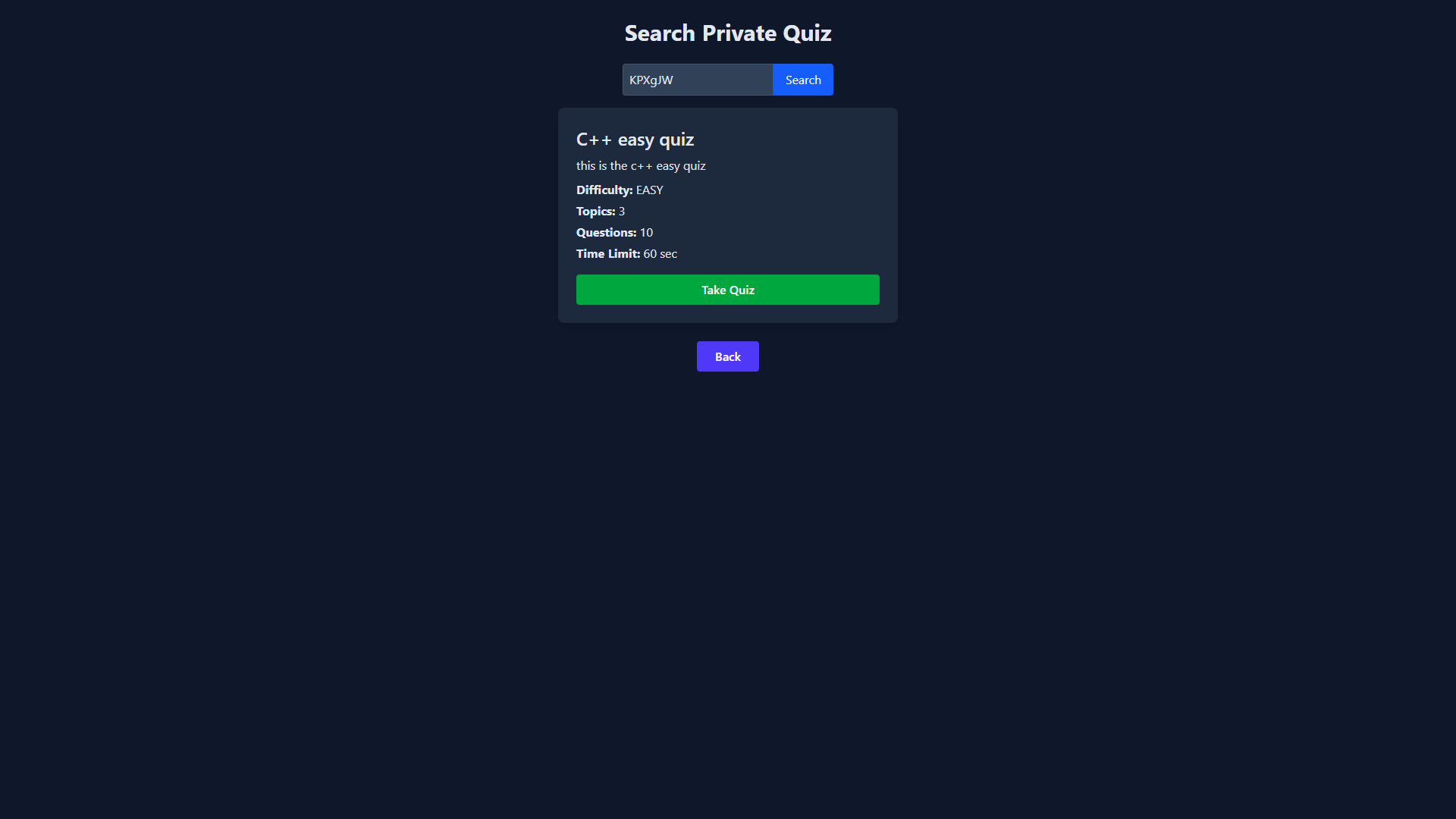
**Filtered quizzes Created By that Student :**

****

**Public Quiz Created by Educator :**

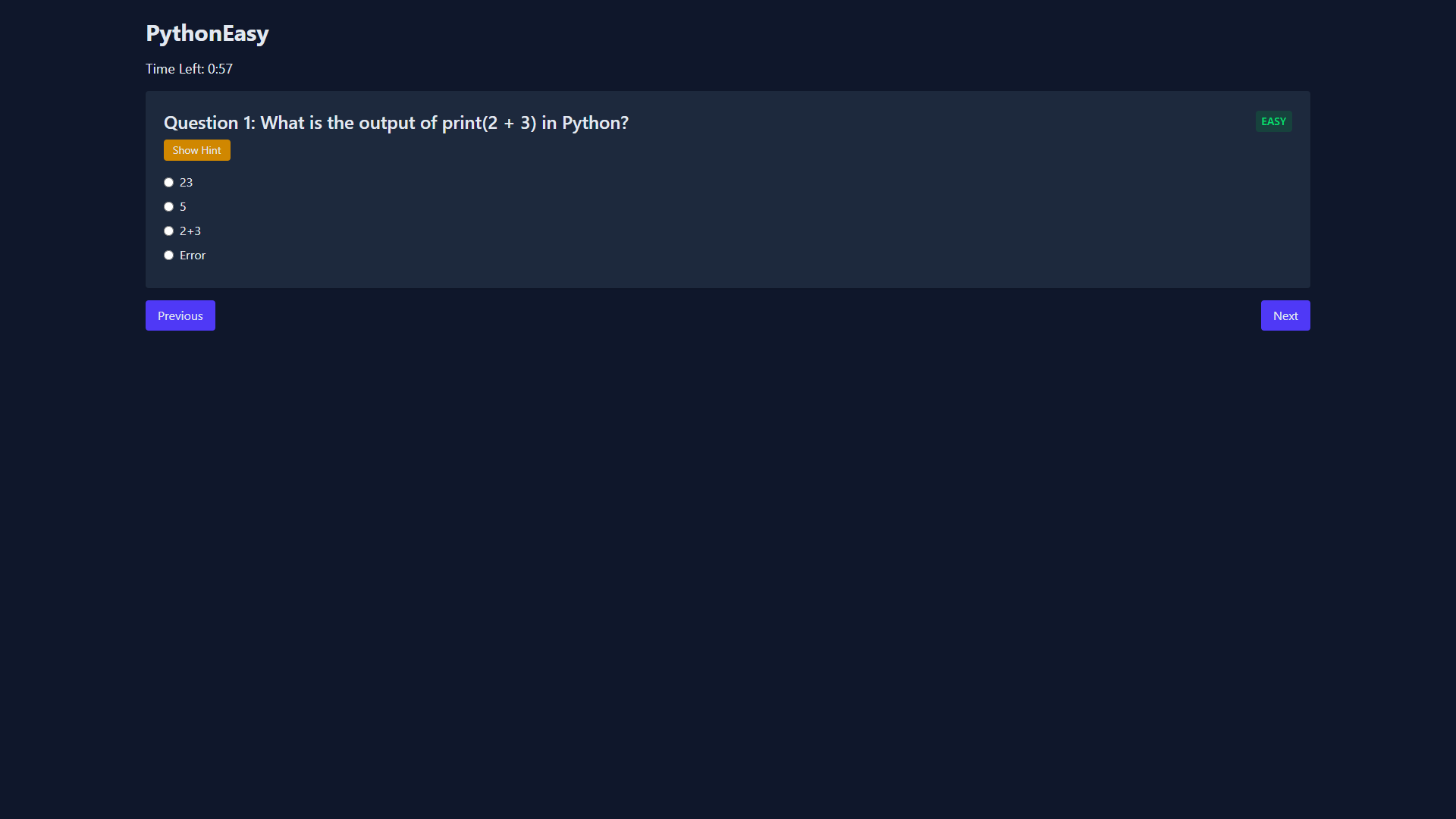
****

**Private Quiz only accessible with the unique code :**

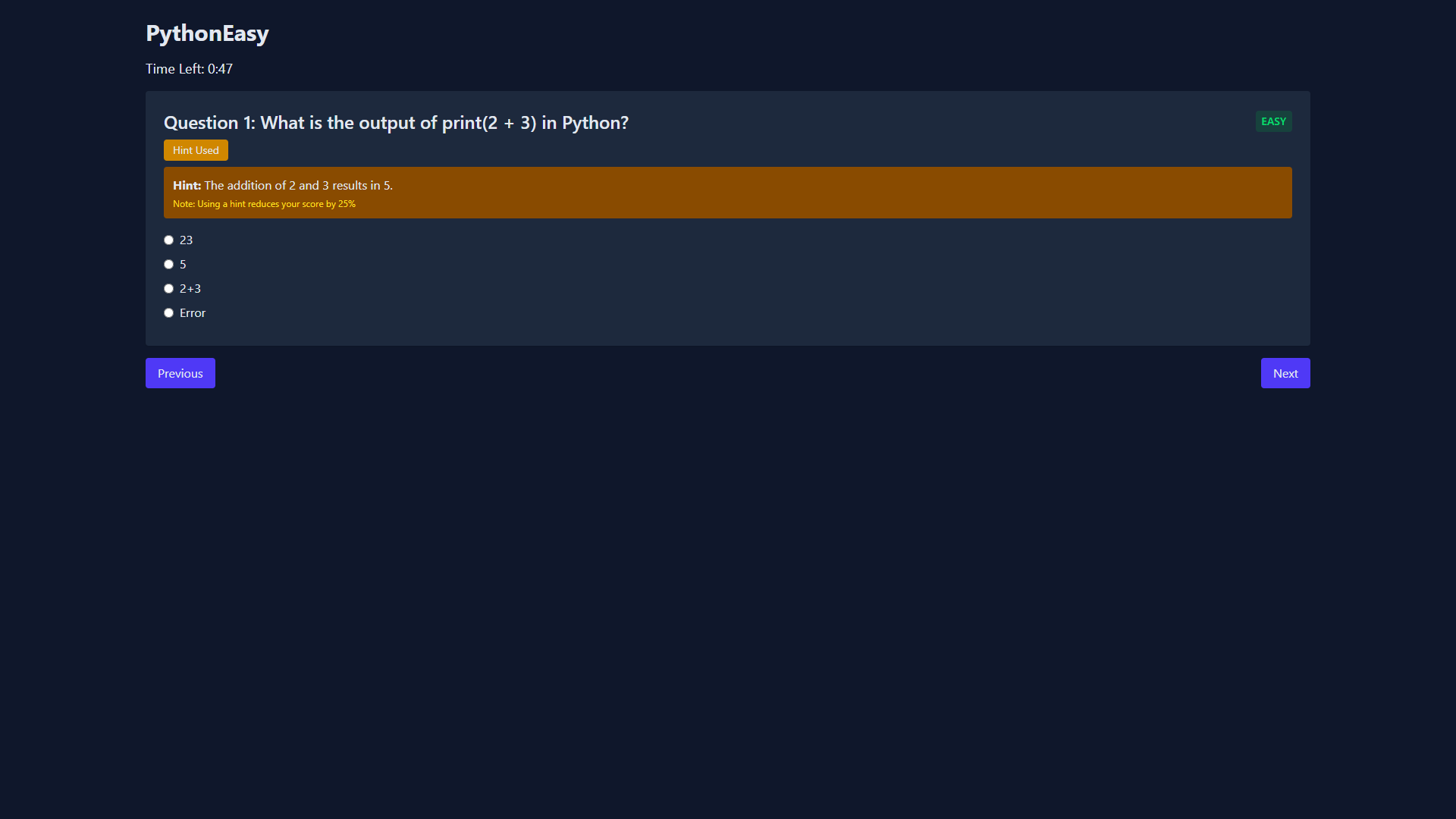
****

**Quiz Attempt :**

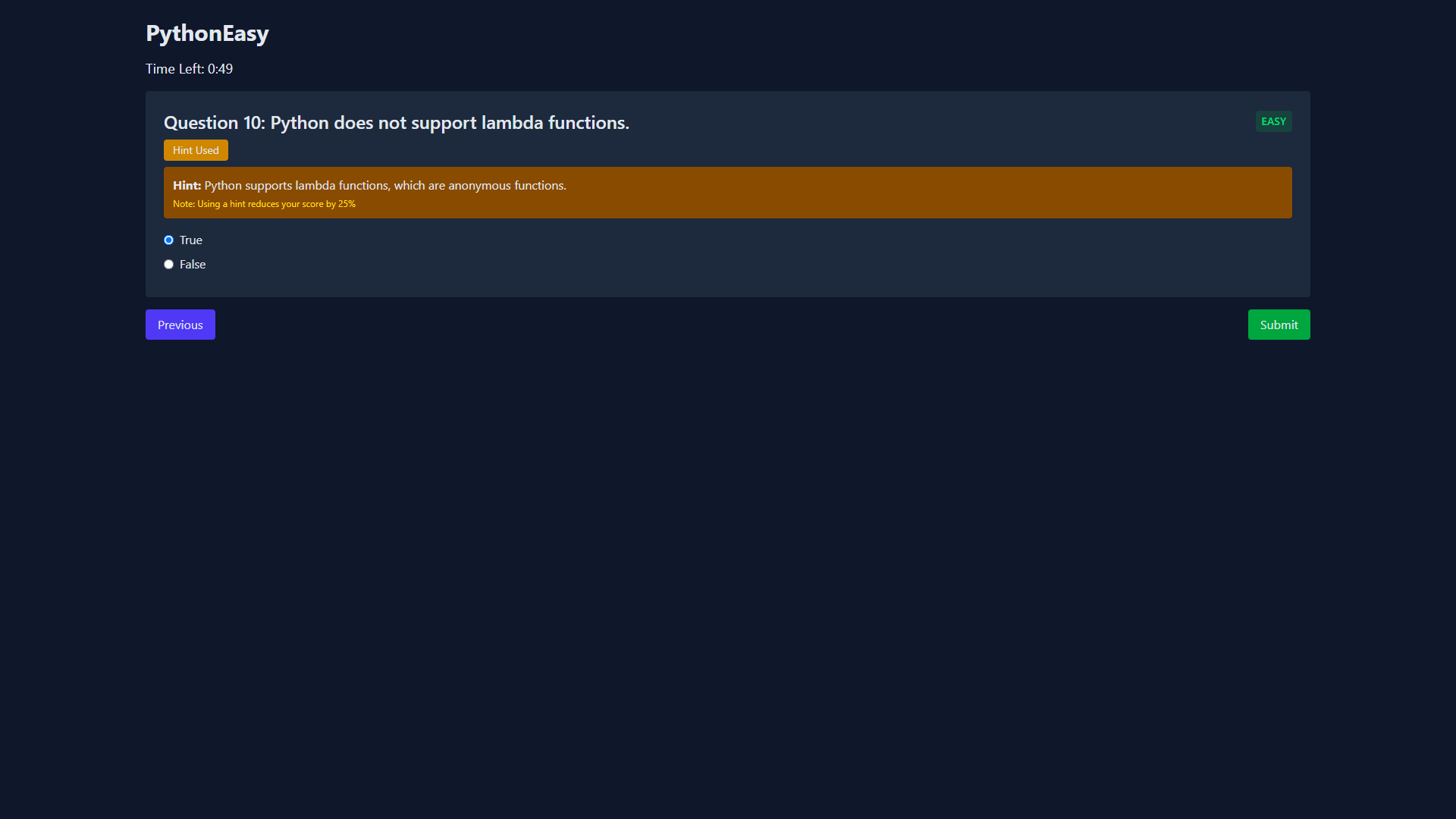
Student can take filtered quiz which can be specified by the user, public which is available to all the users, or private quiz where the educator created for those particular users.

****

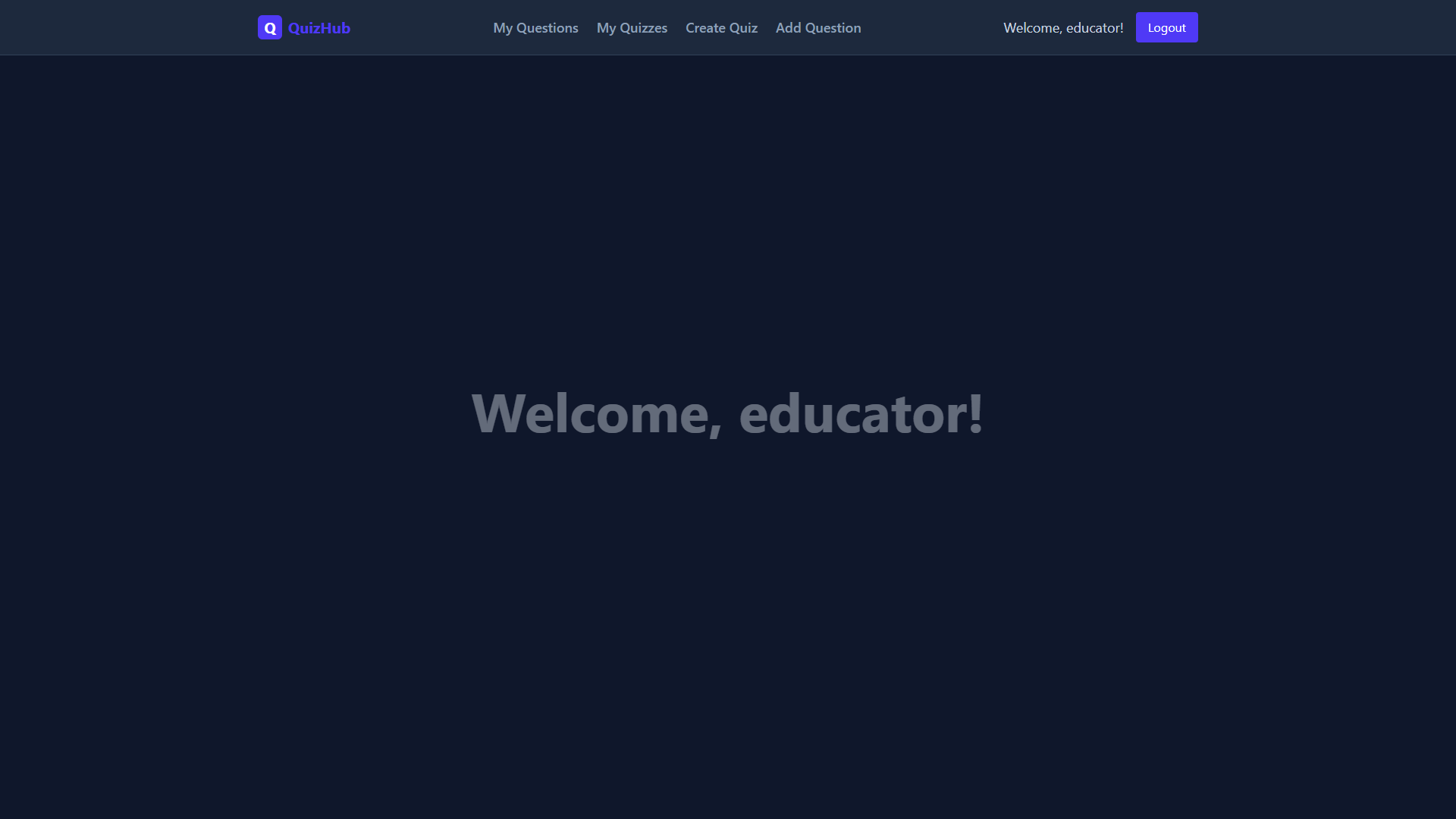
**Using Hint will deduct marks for that question :**

****

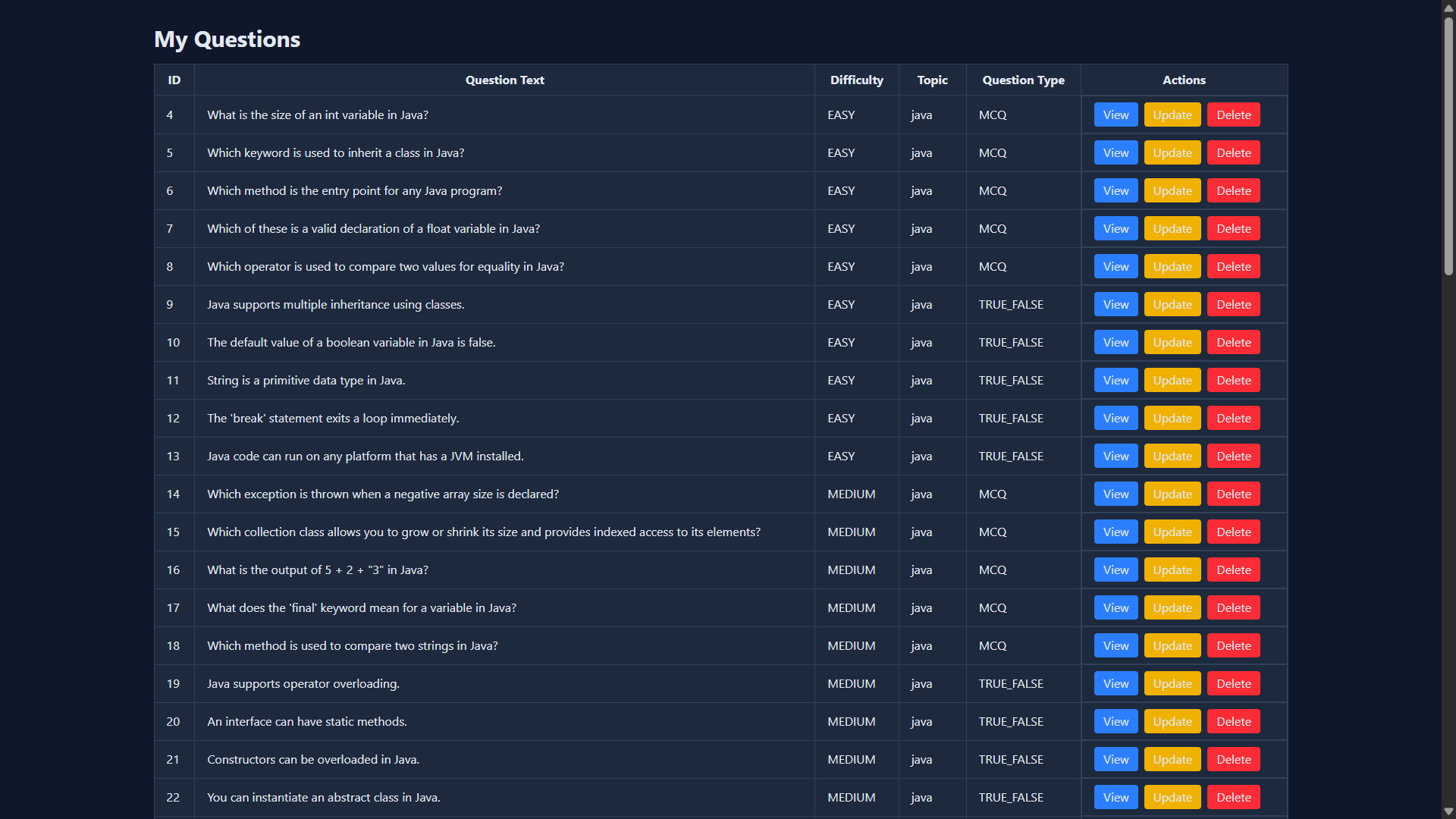
**Submission of Quiz :**

****

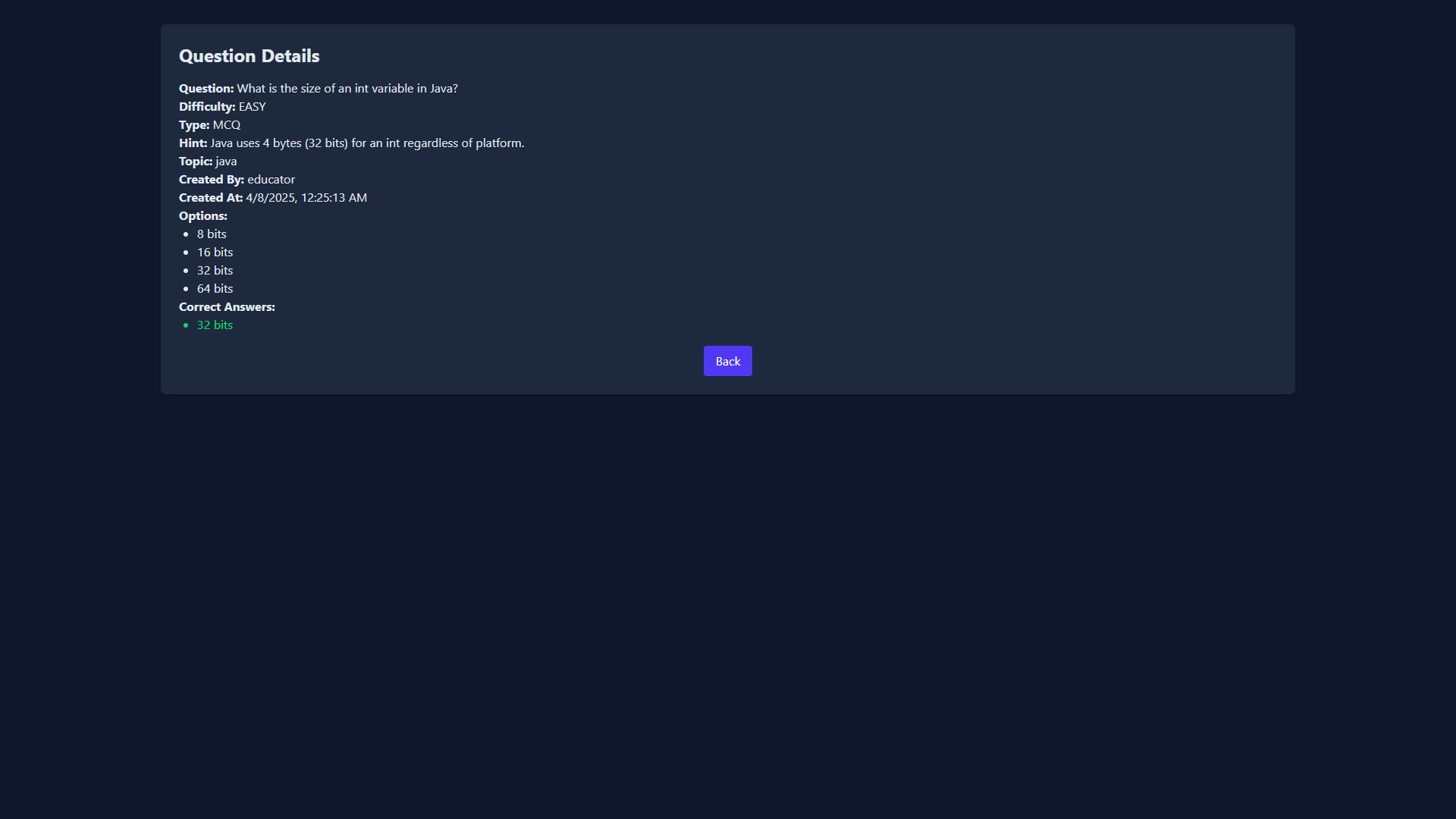
**Educator Dashboard :**

****

**Questions Created by that Educator:**

****

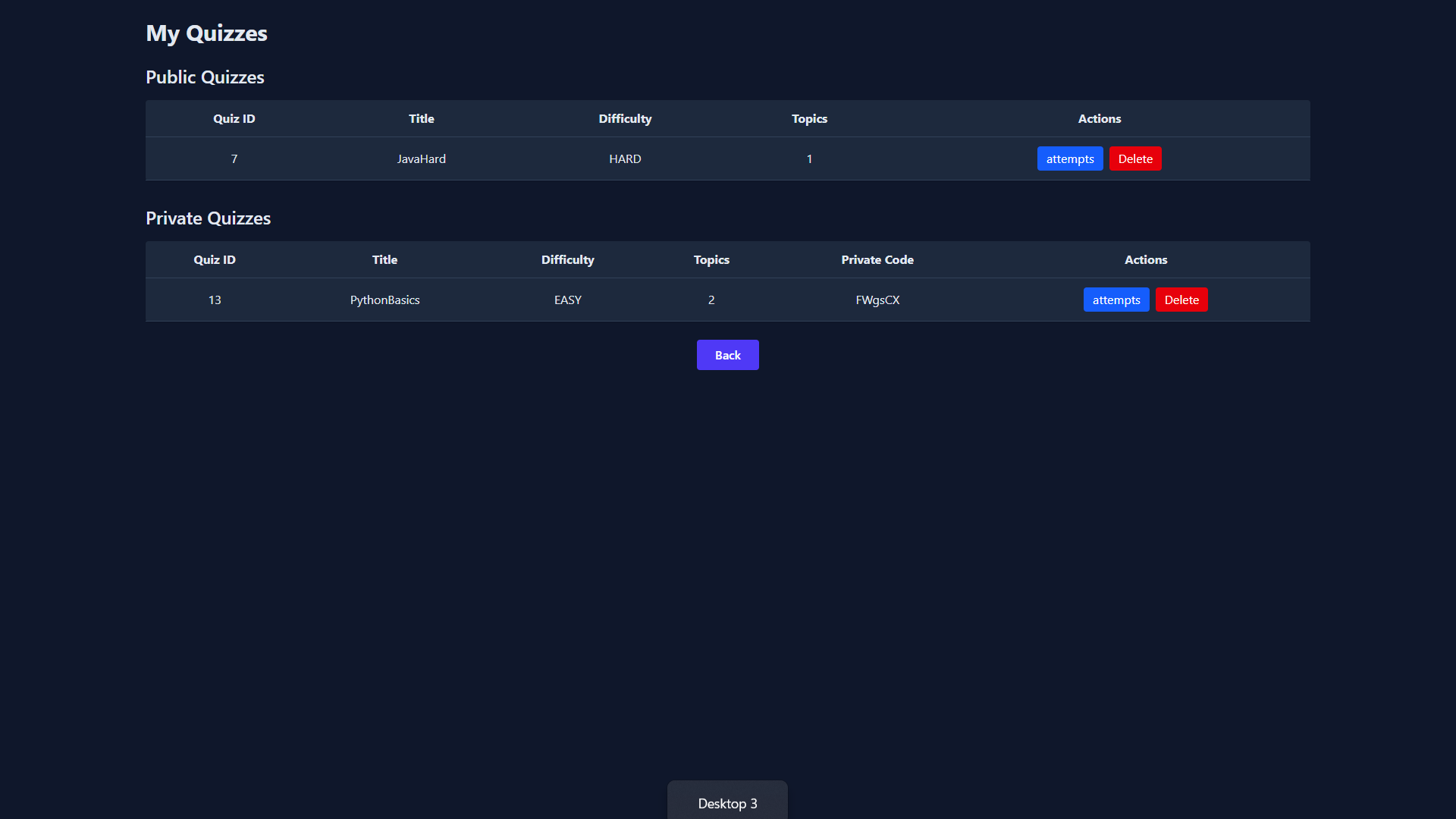
**Viewing Detailed Question:**

****

**Updating the question :**

****

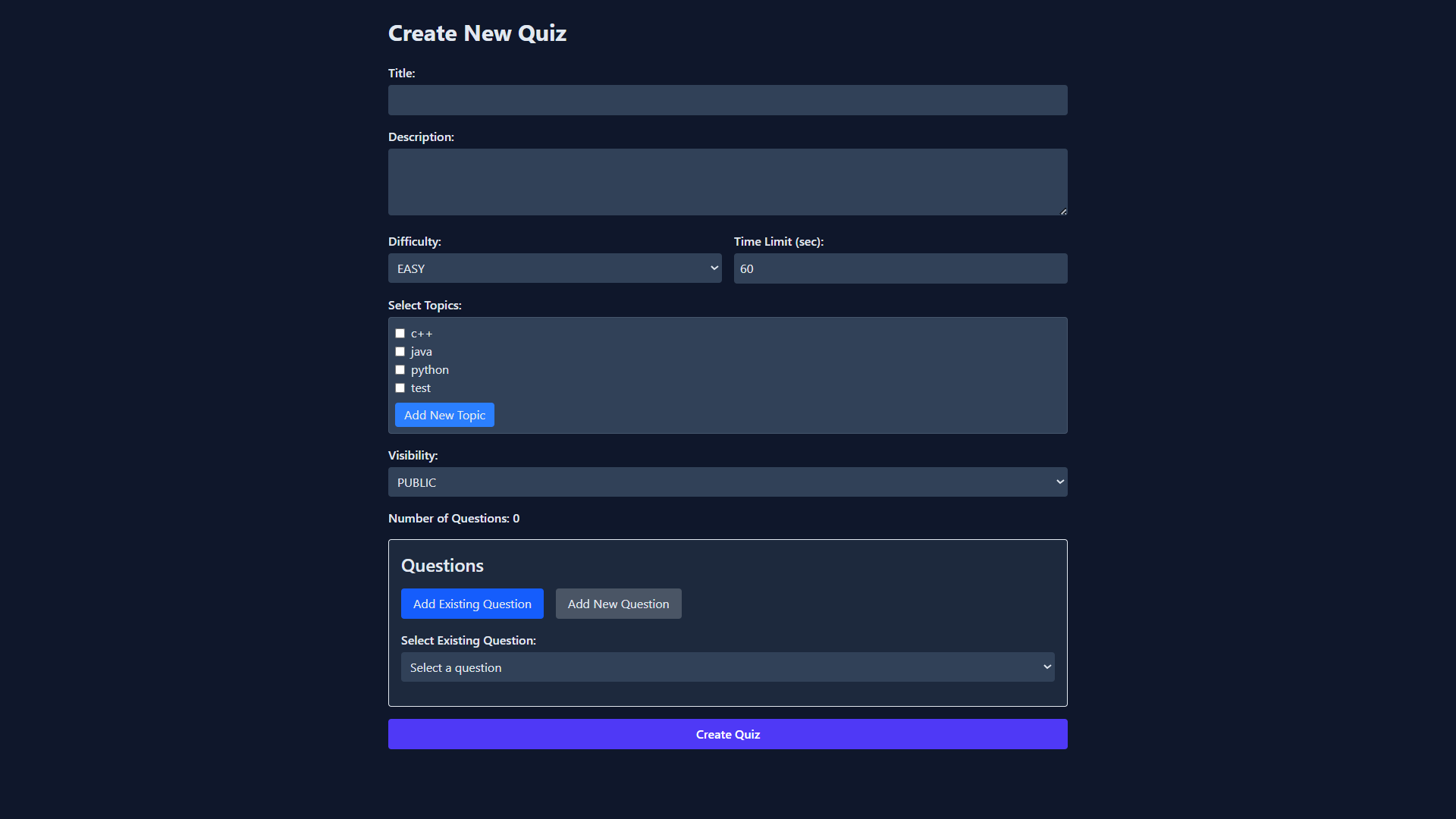
**Educator’s Public and Private Quizzes Created by that Educator:**

****

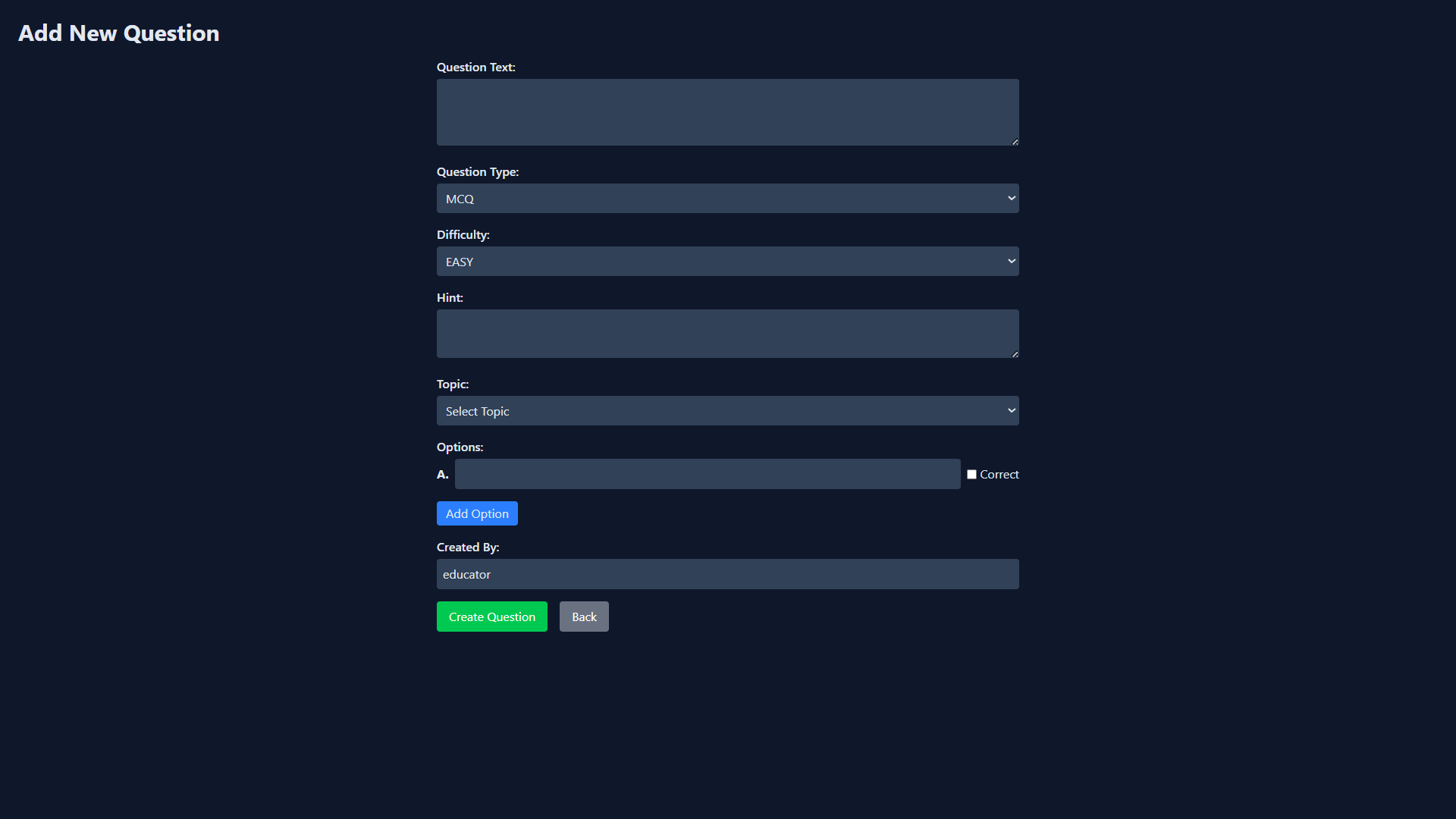
**Attempts of that Quizzes :**

****

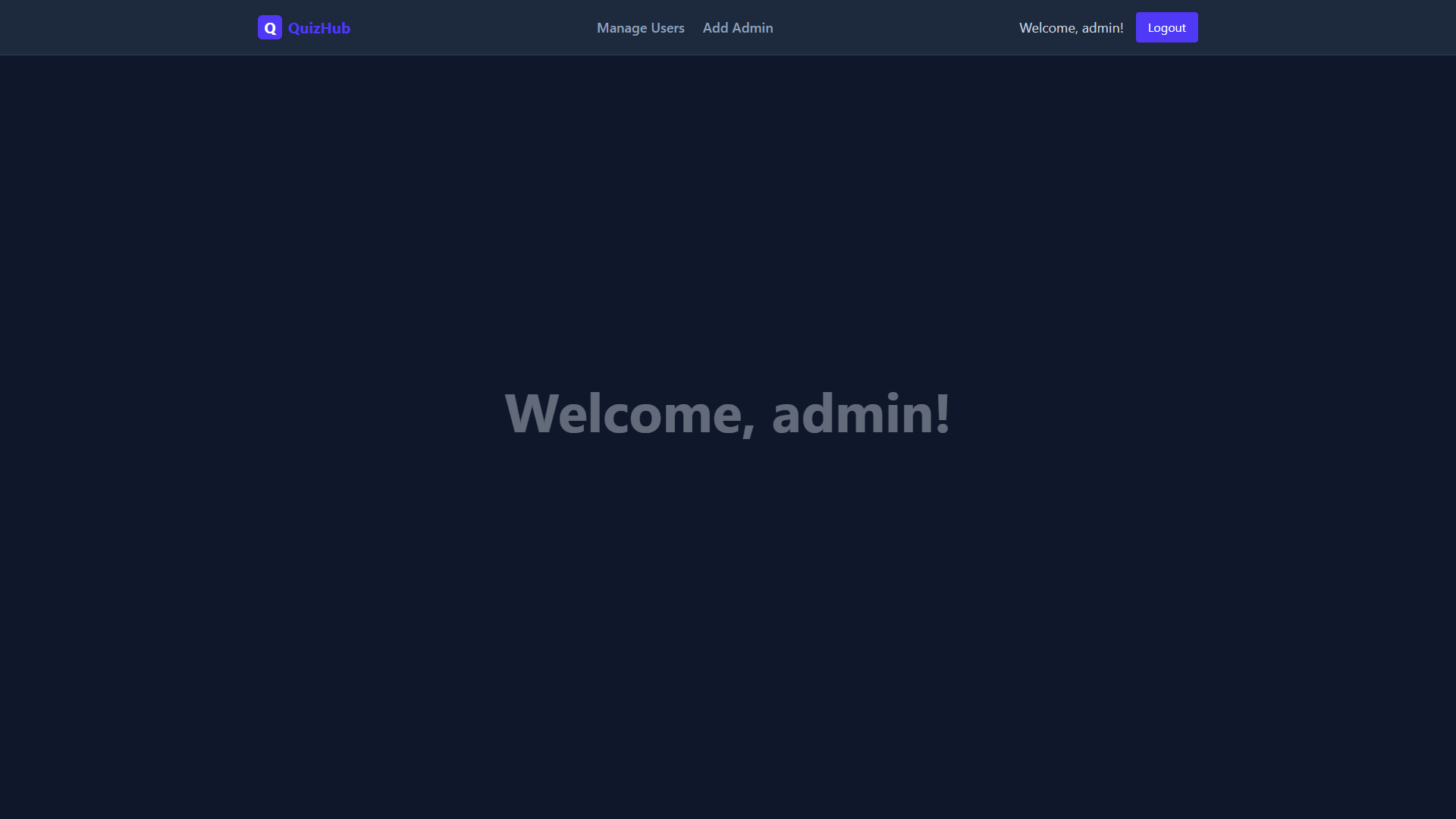
**Quiz Creation :**

****

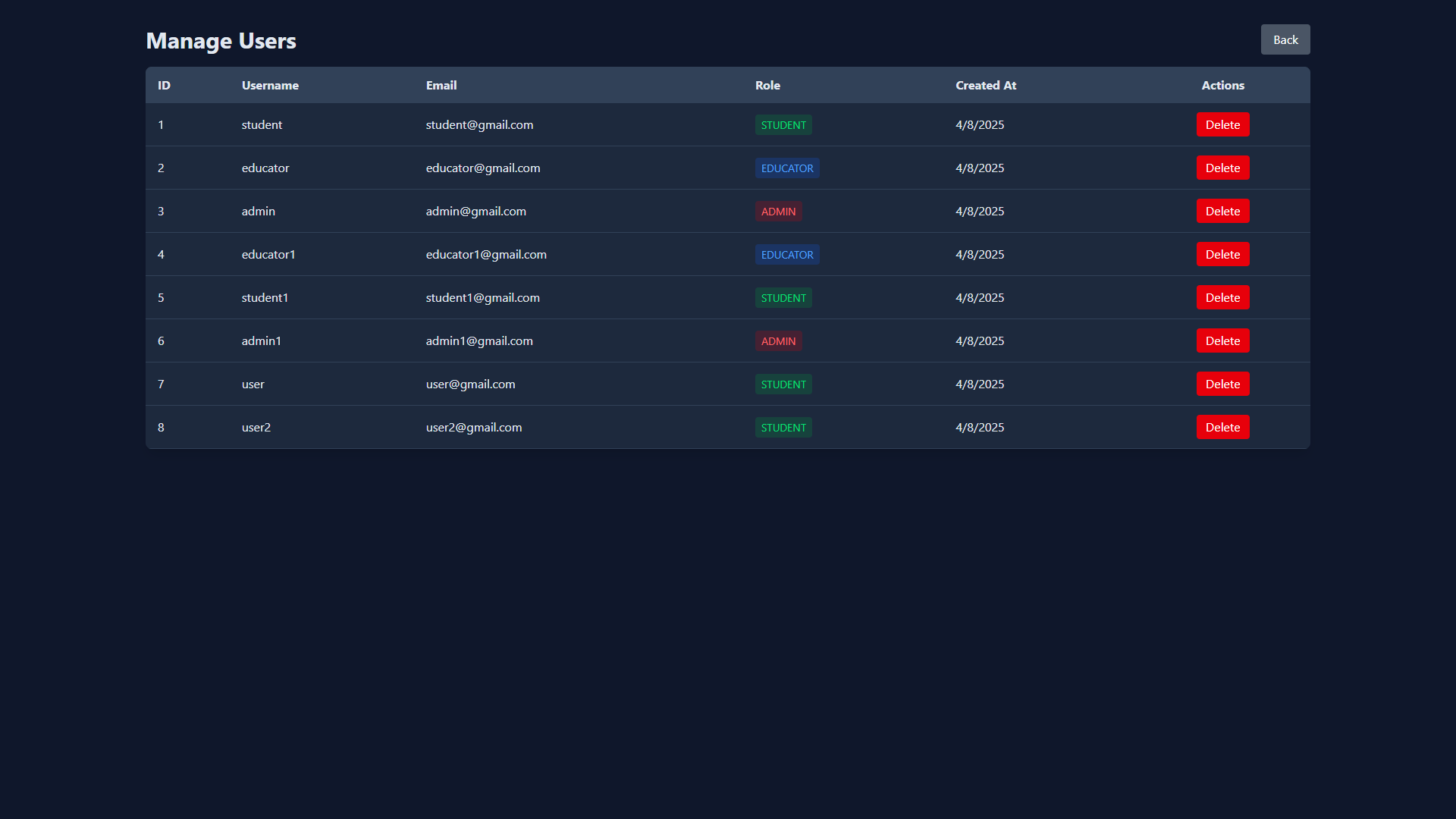
**Adding New Question :**

****

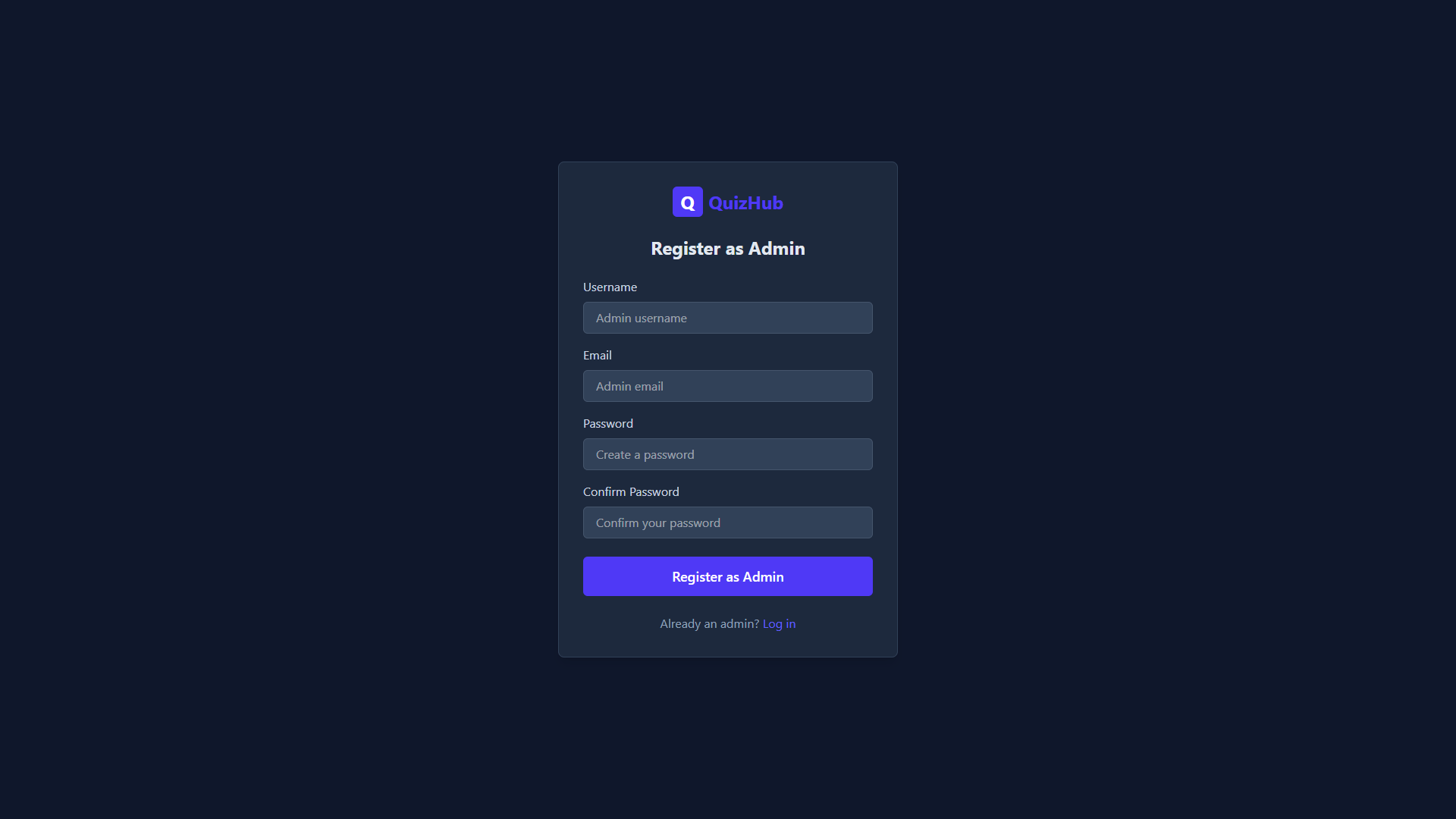
**Admin Dashboard :**

****

**Managing Users :**

****

**Adding New Admin (One Admin can add another Admin):**

****

**Design Patterns Used :**

**Creational Design Patterns:**

**Factory Pattern Implementation in QuizHub**

**What is the Factory Pattern?**

The Factory Pattern is a creation pattern that uses a special method to create objects instead of using the "new" operator directly. This makes it easier to decide which object to create based on different conditions.

**How We Used Factory Pattern in QuizHub**

In our QuizHub application, we need different types of quiz DTOs (Data Transfer Objects) based on quiz properties. We created a factory to handle this logic:

1. **The Factory Class**: QuizDTOFactory has one main method createQuizDTO() that looks at a Quiz object and decides which DTO to create.
2. **Base Class**: BaseQuizDTO contains common properties like title, description, and difficulty.
3. **Specialized Classes**:
   * PrivateQuizDTO: For private quizzes (adds a private code)
   * FilteredQuizDTO: For quizzes with filter options
   * PublicQuizDTO: For standard public quizzes

**Benefits of This Approach**

* We hide the decision-making logic about which object to create
* Code using these DTOs doesn't need to know the specific type
* We can easily add new DTO types in the future
* It's easy to change the creation logic without changing other parts of the code

**How It Works**

When we need a DTO, we simply call:

BaseQuizDTO dto = QuizDTOFactory.createQuizDTO(quizObject);

The factory then checks:

* If the quiz is private → returns a PrivateQuizDTO
* If the quiz has filter criteria → returns a FilteredQuizDTO
* Otherwise → returns a PublicQuizDTO

****

**2. Behavioral Design Patterns**

**Strategy Pattern Implementation in QuizHub**

**What is the Strategy Pattern?**

The Strategy Pattern is a behavioral design pattern that lets you choose different algorithms (strategies) at runtime. It works by defining a family of algorithms, putting each one in a separate class, and making their objects interchangeable.

**How We Used Strategy Pattern in QuizHub**

In our QuizHub application, we need to calculate scores differently based on requirements. The Strategy Pattern helps us switch between scoring algorithms easily:

1. **The Strategy Interface**: ScoringStrategy defines a common method calculateScore() that all scoring strategies must implement.
2. **Concrete Strategies**:
   * DefaultScoringStrategy: Uses standard scoring rules
     + EASY: 1 point
     + MEDIUM: 2 points
     + HARD: 3 points
     + 25% penalty if hint used
   * AdvancedScoringStrategy: Uses enhanced scoring rules
     + EASY: 1.2 points
     + MEDIUM: 2.5 points
     + HARD: 4 points
     + 30% penalty if hint used
3. **Spring Framework Integration**: We use Spring's dependency injection with @Component and @Primary annotations to manage our strategies.

**Benefits of This Approach**

* We can easily swap between scoring algorithms without changing other code
* Adding new scoring methods is simple - just create a new class implementing ScoringStrategy
* Testing becomes easier as each strategy can be tested independently
* The code is more organized and follows the Single Responsibility Principle

**How It Works**

When we need to calculate a score, we simply use the injected strategy:

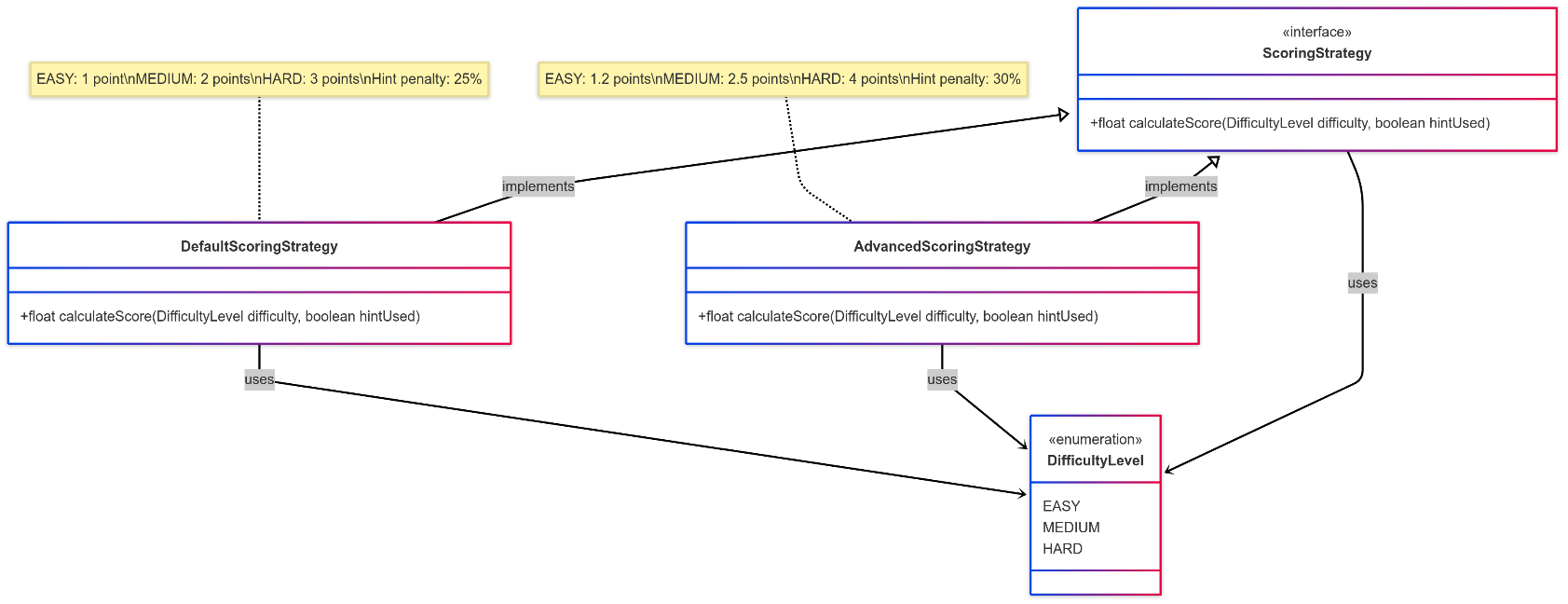
@Autowired

private ScoringStrategy scoringStrategy;

// Later in the code

float score = scoringStrategy.calculateScore(difficulty, hintUsed);

Spring automatically injects the AdvancedScoringStrategy (marked as @Primary), but we can easily switch to another strategy if needed.



**Design Principles :**

**Single Responsibility Principle (SRP)**

**Definition:** Each class should have only one reason to change, meaning it should have only one job or responsibility.

**How Applied in QuizHub:**

* Each controller handles specific operations only (e.g., QuizController only manages quiz operations)
* Service classes are divided by domain (QuestionService, QuizService, etc.)
* Repository classes focus solely on data access

**Example:**

public class QuestionService implements IQuestionService {

// Only handles question-related operations

// Does not manage quizzes or user authentication

}

**Interface Segregation Principle (ISP)**

**Definition:** Clients should not be forced to depend on interfaces they don't use.

**How Applied in QuizHub:**

* The IQuestionService interface contains only methods that clients need

**Example:**

public interface IQuestionService {

Question createQuestion(Question question, Long topicId, Long userId);

Optional<Question> getQuestionById(Long id);

Question updateQuestion(Long id, Question updatedData);

void deleteQuestion(Long id);

List<Question> getQuestionsForUser(String userName);

}

**Open/Closed Principle (OCP)**

**Definition:** Software entities should be open for extension but closed for modification.

**How Applied in QuizHub:**

* Factory Pattern (QuizDTOFactory) allows adding new quiz types without changing existing code
* Adding a new quiz DTO type only requires creating a new class that extends BaseQuizDTO

**Example:**

// To add a new type, we create a new class:

public class SpecialQuizDTO extends BaseQuizDTO {

// New functionality here

}

// The factory remains unchanged but now supports the new type

**Liskov Substitution Principle (LSP)**

**Definition:** Objects of a superclass should be replaceable with objects of its subclasses without breaking the application.

**How Applied in QuizHub:**

* Quiz DTO hierarchy (BaseQuizDTO, PublicQuizDTO, PrivateQuizDTO, FilteredQuizDTO)
* Any method that accepts BaseQuizDTO can work with any of its subtypes

**Example:**

// This works with any quiz DTO type

void processQuiz(BaseQuizDTO quizDTO) {

// Process any quiz DTO (public, private, filtered)

}

**Dependency Inversion Principle (DIP)**

**Definition:** High-level modules should not depend on low-level modules. Both should depend on abstractions.

|  |  |
| --- | --- |
| **Name** | **Module worked on** |
| ABHIMANYU SINGH | **Quiz Attempt** |
| ABHISHEK SHIROL | **Quiz Creation (Public and Private)** |
| ABHISHEK POLICE PATIL | **UserRegistrationAndLogin** |
| ABHIJEET KUMAR | **FilteredQuizCreation** |

**How Applied in QuizHub:**

* Service classes depend on repository interfaces, not concrete implementations
* Strategy pattern uses the ScoringStrategy interface, not concrete scoring classes

**Example:**

@Service

public class QuestionService implements IQuestionService {

@Autowired

private QuestionRepository questionRepository;

// Service depends on the repository interface

}

[**Github link to the Codebase:**](https://github.com/AbhishekShirol/Quiz-Hub)

<https://github.com/AbhishekShirol/Quiz-Hub>