**Project Title: Online Quiz System**

**Submitted by: Abdul Moiz**

**Roll Number: 242337**

**Class: Bscys(eve)-F24B**

**Instructor: Mam Irum Fatima**

****

****

**INTRODUCTION & SYSTEM OVERVIEW**

**1. Abstract**

The *Online Quiz System* is a full-stack learning project built using C++ and web technologies. It enables institutions, instructors, and students to manage and take quizzes online in a secure, interactive manner. The system is designed to mimic real-world exam conditions, allowing teachers to create quizzes and students to take them through a simple web interface.

Built entirely in C++ with a lightweight embedded web server , this system showcases how powerful object-oriented programming (OOP) can be applied outside the console environment. It features real-time cheating detection, automatic result storage, and complete quiz management through file handling — all while maintaining a responsive GUI using HTML and CSS.

This project is not only a demonstration of technical proficiency but also of system design thinking, user experience consideration, and secure logic enforcement — making it a powerful example of bridging classic system programming with modern web-based interfaces.

**2. Objectives**

* Develop a complete online examination platform using **pure C++**, HTML, and CSS, without relying on databases.
* Explore and apply core OOP concepts like **inheritance**, **polymorphism**, **encapsulation**, and **abstraction** by mapping real-world users and entities to C++ classes.
* Demonstrate the integration of **file I/O for data persistence**, using cleanly structured text files to simulate a real database.
* Build an **interactive GUI** with dynamic data binding, template rendering, and user-triggered navigation through web pages served by the C++ backend.
* Introduce basic **security measures** like encrypted password storage and disqualification mechanisms to improve the integrity of online assessments.
* Emphasize **modularity and scalability**, allowing more features (like timers, leaderboards, analytics) to be added later with minimal refactoring.

**3. Scope of the Project**

The Online Quiz System was developed to simulate a modern-day examination and learning environment within the constraints of file-based storage and low-dependency systems. Its primary scope includes:

* **Educational Institutions**: Schools, colleges, and coaching centers where instructors need a digital platform for conducting secure assessments.
* **Students**: Individuals who want to attempt quizzes independently in a focused environment, without needing third-party tools or a full database server.
* **Teachers**: Educators looking to create, manage, and evaluate multiple-choice quizzes through a simple interface.
* **Local/Offline Testing**: Especially in environments with limited or no internet access, this system can run entirely on localhost.

This system has been purposefully designed to be simple yet effective — avoiding dependencies like SQL or full-stack frameworks — making it ideal for demonstrations, offline use, labs, and learning environments.

**4. System Functionality**

**User Roles:**

* **Student**: Register/Login, take quiz, get disqualified if they leave the tab, view results.
* **Teacher**: Register/Login, create quizzes, view student results.

**Core Actions:**

* Register or log in using the GUI
* Teachers can create quizzes with multiple-choice questions saved to files
* Students select quizzes and answer questions with real-time validation
* If a student switches tabs, they are disqualified immediately
* Final scores are stored in a result file with timestamps

**5. Features of the Online Quiz System**

**User Roles:**

* **Student**:
  + Register/login via GUI
  + Select and take quizzes
  + Automatically disqualified if switching tabs
  + View personal result history
  + One attempt per quiz (enforced)
* **Teacher**:
  + Register/login via GUI
  + Create new quizzes by entering questions and options
  + View results submitted by students

**Quiz Management:**

* Quizzes are created and stored as **text files** (data/quizname.txt)
* Each question includes:
  + A question statement
  + Four options (A–D)
  + Correct option index
* Quizzes are loaded dynamically when students begin a test

**Security & Novelty Features:**

* **Tab switch disqualification**
* **Role-based file-level login**
* **One-attempt-per-quiz enforcement**
* Passwords are stored with basic XOR encryption for learning purpose
* No external database — everything is done with file I/O

**6. Technical Stack**

| **Layer** | **Technology Used** |
| --- | --- |
| Language | C++ (OOP) |
| GUI | HTML + CSS |
| Storage | Text files (users.txt, results.txt, quizzes.txt) |
| Encryption | XOR-based string encryption (basic learning-level) |
| Anti-Cheat | JavaScript + visibilitychange listener |

**7. Implementation Highlights**

**File-Based Storage**

* All user data is stored in users.txt in the format:
* Quizzes are saved as quiz\_title.txt, with each question and its four options followed by the correct answer index.
* Results are stored in results.txt in the format

**Page Routing with httplib**

* Routes like /, /login, /student, /teacher, /student/start, /student/next, /student/results are defined in main.cpp using lambda functions.
* Dynamic HTML templates (.html files) are loaded, modified using placeholder replacement, and sent to the browser.

**Placeholder Replacement Logic**

* HTML files contain tags like {{question\_text}}, {{quiz}}, {{score}}, which are replaced with real data before being served.
* This is handled using a helper function replace\_all() that finds and replaces all instances of a placeholder in a string.

**Folder Structure:**

**project**

**src**

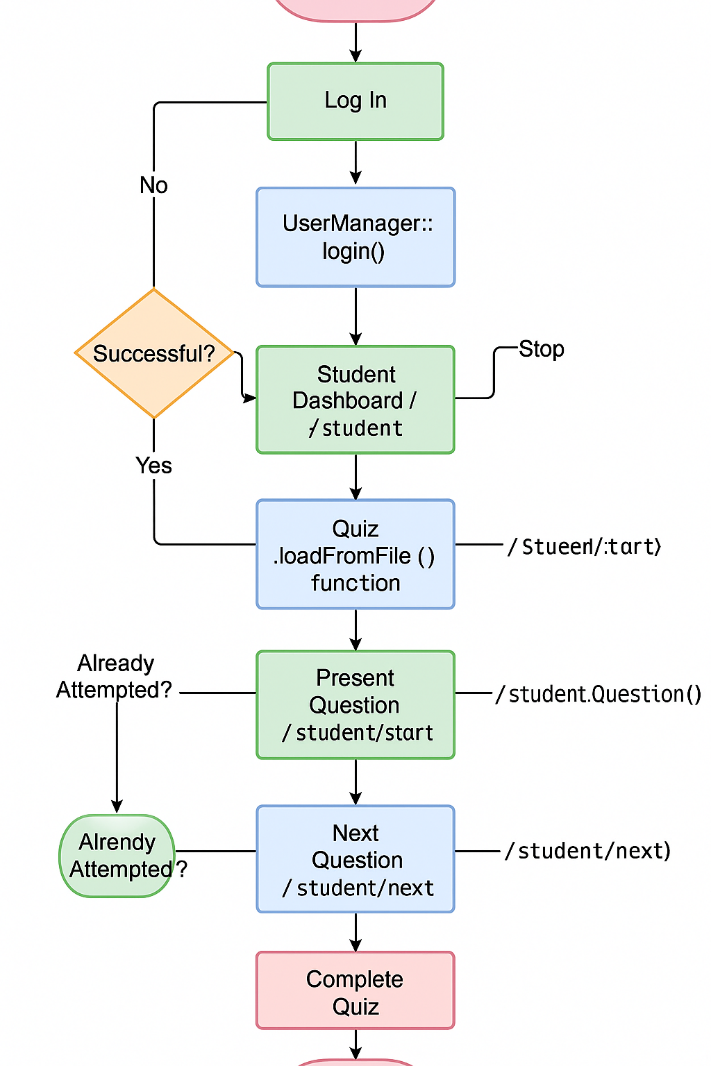
* Quiz.cpp / .h
* Question.cpp / .h
* Student.cpp / .h
* Teacher.cpp / .h
* UserManager.cpp / .h
* FileManager.cpp / .h
* Utils.cpp / .h

**web**

* index.html
* quiz\_question\_template.html
* student\_results\_template.html
* style.css

**data**

* users.txt
* results.txt
* [quizname].txt



**8. Novelty & Innovation**

What makes this project stand out is the way it combines **modern web interaction** with a **low-level language like C++**, often considered unsuitable for GUIs and web servers. Key innovations include:

* **Tab-Switch Disqualification**: Using the visibilitychange event in JavaScript, the system monitors if the student moves to another tab or window. If detected, it auto-submits a zero score and prevents further cheating — mimicking real proctoring environments.
* **CLI-to-GUI Transformation**: Originally a command-line quiz, this project has been evolved into a full HTML GUI without using frameworks like React or PHP. Instead, HTML templates are dynamically served from within C++ using string manipulation.
* **No Database Used**: Every quiz, result, and user is stored in text files with custom structure — demonstrating how real-world systems can still be prototyped and maintained without SQL.
* **One-Attempt Rule Enforcement**: The backend checks if the student already attempted the quiz, ensuring fairness and protecting the test's integrity.
* **Full Role-Based Flow**: Teachers and students have entirely separate logic and views — handled using one core C++ server.

This project pushes C++ usage into the realm of full-stack development in a clean and understandable way — something rarely seen at the student level.

**9. Challenges & Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| Integrating C++ with a GUI | Used httplib.h to simulate a web server and served HTML/CSS from it. |
| Preventing multiple quiz attempts | Implemented a check inside /student/start route that looks into results.txt before letting a student proceed. |
| Keeping HTML dynamic without a templating engine | Used a custom replace\_all() C++ function to inject data into HTML files. |
| Validating browser behavior like tab-switching | Used document.visibilitychange in JavaScript to detect loss of focus and disqualify. |
| Making the system platform-independent | Avoided GUI libraries like Qt or Windows Forms in favor of HTML/CSS, which works on any OS. |

Each challenge was approached with **modular thinking**, allowing the system to stay small, simple, and efficient.

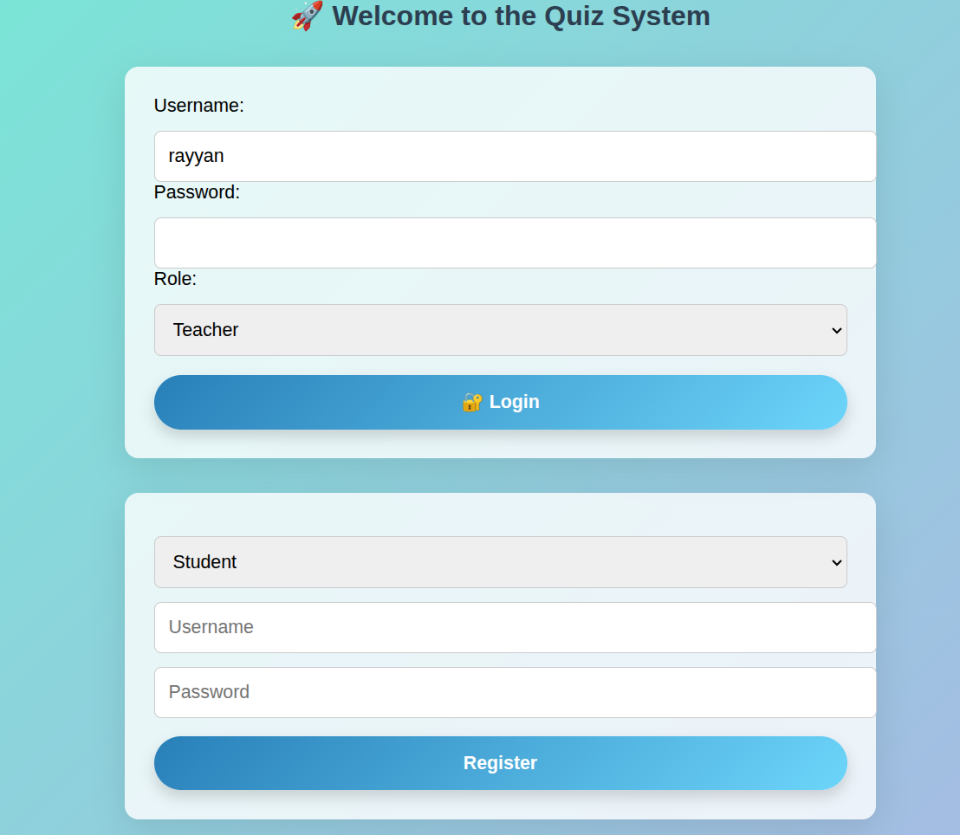
**10. Future Enhancements**

Some planned or suggested future upgrades include:

* **Real-Time Timer**: Add a countdown timer per quiz. If time runs out, auto-submit the result.
* **Question Randomization**: Shuffle question order for each student to reduce chances of cheating.
* **Student Leaderboard**: Sort students by score across all quizzes and display a top-10 list.
* **Email Notifications**: Send quiz results to students via email using a simple SMTP C++ library.
* **Admin Role**: Introduce an admin role who can manage users, reset passwords, or delete quizzes.
* **Quiz Analytics Dashboard**: Show stats like average score, most missed question, difficulty level.
* **Database Integration**: Replace file I/O with SQLite or MySQL for larger-scale use cases.
* **Mobile-Friendly Frontend**: Use responsive design so quizzes can be taken on phones or tablets.

**11. Interface**

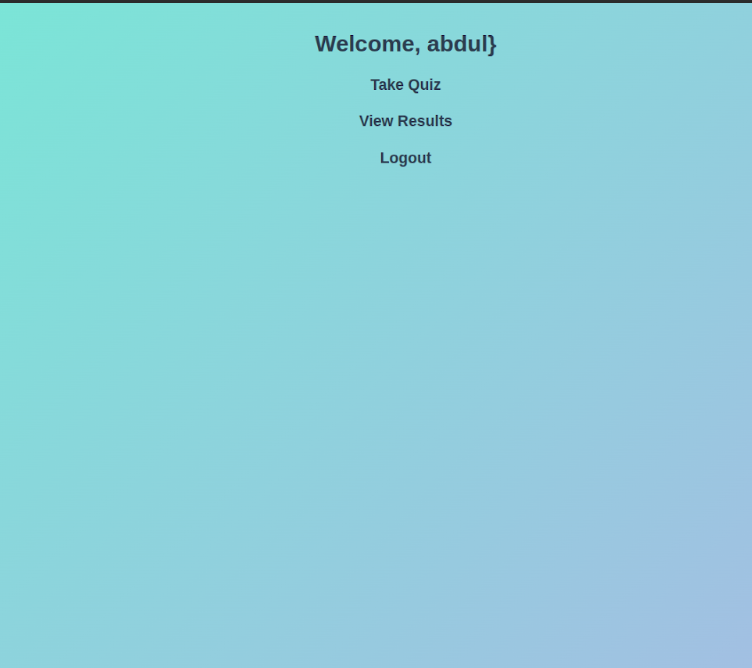
Login page:



Login as teacher



Login as student



**12. Conclusion**

The *Online Quiz System* is a well-rounded, robust project that transforms classic C++ programming into a full user-interactive application with web capabilities. It demonstrates not just coding skill but an understanding of how to create systems that are **usable**, **secure**, and **scalable**.

By combining CLI logic with web interaction, enforcing anti-cheat logic, and following a modular class design, this system goes far beyond a typical school-level project. It shows how even without external databases or frameworks, a thoughtful developer can craft a reliable, extensible, and professional-grade application.

This project is both a strong academic submission and a demonstration of industry-relevant design principles — making it an ideal highlight in a portfolio or resume.