MaswaliYote Quiz Management System (MQMS) - Analysis Report

This analysis report provides a snapshot of MaswaliYote Quiz System's current state and suggests areas for future consideration and enhancement.

Introduction:

MQMS is a console-based application designed to facilitate quiz creation, management, and participation. It caters to two main user roles: administrators who create/manage quizzes, and students who take quizzes and view their results.

MQMS Features:

1. User Authentication:

- Admins and students can securely log in with their usernames and passwords.

2. Admin Functionality:

- Admins can create quizzes with multiple-choice questions.
- Admins can view, update, and delete quizzes.
- Admins can manage user accounts, including creation, viewing, updating, and deletion.

3. Student Functionality:

- Students can answer quizzes with multiple-choice questions.
- Students can view their quiz results.

4. Security:

- User passwords are hashed and stored securely.
- Sensitive data is protected from unauthorized access.

5. Usability:

- The system provides a clear and intuitive console-based interface.

6. Reliability:

- The system handles errors gracefully.

7. Scalability:

- The system is designed to handle a growing number of quizzes and users.

Checklist:

A. Functional Requirements:

a. User Authentication implemented.
b. Admins can create quizzes.
c. Admins can view, update, and delete quizzes.
d. Admins can manage user accounts.
e. Students can answer quizzes.
f. Students can view quiz results.

B. Non-functional Requirements:

a. User passwords are hashed.
b. The console interface is clear and intuitive.
c. The system handles errors gracefully.
d. The system is designed for scalability.

C. Security:

a. User passwords are securely hashed.
b. Access controls are in place for admin and student functionality.
c. Sensitive data is protected from unauthorized access.

D. Usability:

a. The console-based interface is user-friendly.
b. Instructions are clear for admins and students.

E. Reliability:

a. Error handling is implemented.
b. The system operates reliably under normal usage.

F. Scalability:

a. The system is designed to handle a growing number of quizzes and users.

G. Code Quality:

a. Code follows best practices.
b. Code is modular and maintainable.
c. Documentation is provided.

H. Testing:

a. Unit tests cover critical functionalities.
b. Integration tests validate system behavior.
c. Edge cases are considered in testing.

I. Collaboration:

a. Code is version-controlled (e.g., Git).
b. Clear contribution guidelines are provided.
c. Proper commit messages are used.

J. Licensing:

a. A clear license (e.g., MIT) is specified.
b. License terms are respected.

Conclusion:

The MQMS successfully meets its functional and non-functional requirements. It demonstrates good security practices, usability, reliability, and scalability. The codebase is well-organized and follows best practices. Proper testing and version control contribute to a robust and collaborative development environment.

Recommendations:

1. Consider adding more comprehensive testing, including edge cases.
2. Regularly update dependencies for security and feature enhancements.
3. Provide continuous documentation updates for maintainability.
4. Encourage a collaborative community by actively engaging contributors.
5. Consider incorporating user feedback for further improvements.