International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Online Quiz Management System

Sagar Kumar Behera
Students, Dept. of CSE (AI), GIFT Autonomous, Bhubaneswar

Asst. Prof. Tarun Kumar
Assistant Professor, Dept. of CSE (AI), GIFT Autonomous, Bhubaneswar

***Abstract*—The Online Quiz Management System is a web-based application designed to simplify and automate the process of conducting online quizzes and examinations. The system provides an efficient platform for administrators to create, manage, and evaluate quizzes, while allowing students to participate in examinations through a user-friendly interface. The project aims to reduce manual work, improve accuracy in result generation, and provide a fast and secure online examination environment.**

The platform is developed using modern web technologies: HTML and CSS design the frontend interface; JavaScript adds interactivity and client-side validation; PHP handles server-side processing; and MySQL manages the relational database. The system supports secure user registration and login, role-based access control, quiz management, automatic answer evaluation, and instant result generation.

The application supports multiple-choice question-based examinations with timer-based quizzes, automatic score calculation, and database-backed storage of user profiles, quiz data, and attempt records. Administrators manage questions, users, and quiz configurations efficiently, while students can attend assessments from any internet-connected device at any time.

The system aims to improve the efficiency of examination management, minimize paperwork, reduce human errors, and provide a reliable digital assessment platform for educational institutions and training environments.

***Keywords*—Online Quiz, Web Application, MySQL, PHP, MCQ, Timer, Result Generation, JWT Authentication, RBAC, Educational Technology**

I. INTRODUCTION

The rapid advancement of information technology has profoundly transformed education and assessment systems worldwide. Traditional pen-and-paper examinations, once the standard for evaluating student performance, are increasingly being replaced by digital alternatives due to their inherent inefficiencies: they require significant preparation time, manual grading effort, and are prone to human errors in evaluation and data recording. As institutions scale and student populations grow, these limitations become critical bottlenecks in the educational process.

Online assessment systems have emerged as a powerful solution, offering automation, scalability, and instant feedback. The Online Quiz Management System developed in this paper is a web-based application that automates the entire quiz lifecycle — from creation and administration to evaluation and reporting. Administrators can design quizzes with multiple-choice questions, assign time limits, manage user accounts, and monitor performance across the student population. Students can log in from any location, select available quizzes, attempt them within the allotted time, and receive their results immediately upon submission.

The primary motivation for this project is to address the critical pain points of traditional examination management: manual answer checking, delayed result publication, physical logistics of question paper distribution, and the inability to analyze performance data efficiently. By automating these processes, the system frees educators to focus on teaching rather than administrative tasks, and provides students with faster, more transparent feedback on their performance.

Beyond the immediate operational benefits, the system also demonstrates the practical application of modern web development technologies — HTML5, CSS3, JavaScript, React.js, PHP, Node.js, and MySQL — in building a real-world educational tool. The architecture is designed with security, scalability, and extensibility in mind, providing a foundation that can be enhanced with additional features such as AI-based question generation, adaptive testing, and learning analytics in future iterations.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

The remainder of this paper is organized as follows: Section II presents a literature survey of existing online quiz systems and identifies research gaps. Section III describes the system design and architecture. Section IV details the technology stack. Section V covers implementation specifics of each module. Section VI presents testing strategy and results. Section VII discusses findings and limitations. Section VIII concludes with directions for future work.

II. LITERATURE SURVEY

A. Overview of Online Quiz Systems

Online quiz systems are web-based or application-based platforms that conduct assessments digitally through the internet. They allow participants to engage remotely using computers, tablets, or smartphones, eliminating geographic and logistical barriers. Since the early 2000s, such systems have evolved from simple static HTML forms to sophisticated platforms supporting multimedia questions, adaptive difficulty, real-time analytics, and AI-driven evaluation. Their adoption has accelerated significantly in the post-pandemic era, where remote learning became a necessity rather than an option.

Key advantages of online quiz systems over traditional methods include instant and automated grading, time-bound assessments with automatic enforcement, real-time performance feedback, centralized data storage enabling longitudinal analysis, reduced use of paper and physical resources, and the ability to serve thousands of simultaneous users across geographic boundaries. These properties make online quiz systems essential infrastructure for modern educational institutions and corporate training programs.

B. Analysis of Existing Platforms

Google Forms is widely used for basic quiz creation due to its zero-cost availability and integration with Google Workspace. However, it provides only basic grading (auto-grade for MCQs), lacks detailed analytics, does not support time-limited quizzes natively, and provides no role-based access system — making it unsuitable for formal examinations.

Kahoot and Quizizz focus on gamified learning experiences, introducing leaderboards, music, and competitive elements. While effective for informal classroom engagement, they lack examination-grade security controls, custom branding, institutional user management, and detailed post-assessment analytics. Quizizz supports homework mode with time limits, but its quiz creation interface offers limited flexibility for institutions with specific question formats.

Moodle is the most feature-complete open-source Learning Management System, with a powerful quiz module supporting various question types, detailed reporting, and integration with institutional grade books. However, its complexity demands considerable technical expertise for installation, configuration, and maintenance, making it impractical for smaller institutions or organizations without dedicated IT staff. Additionally, Moodle's monolithic architecture can present scalability challenges.

ProProfs Quiz Maker and ExamSoft are commercial platforms offering advanced proctoring and analytics, but involve significant licensing costs, limiting their accessibility for public educational institutions in developing regions. Table I summarizes the feature comparison:

Feature	Google Forms	Kahoot	Moodle	Proposed
Custom User Roles	No	No	Yes	Yes
Timer-Based Quiz	No	Yes	Yes	Yes
Instant Results	Yes	Yes	Yes	Yes
Admin Dashboard	Limited	Limited	Full	Full
JWT Security	No	No	Partial	Yes
Free/Open Source	Yes	No	Yes	Yes
MCQ Support	Yes	Yes	Yes	Yes
Custom Branding	No	No	Yes	Yes
Score Analytics	Basic	Basic	Advanced	Moderate
Mobile Support	Yes	Yes	Yes	Planned

Table I: Feature Comparison of Quiz Platforms

C. Research Gap

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

The review reveals several persistent gaps: (1) Most free platforms lack robust, institution-specific customization. (2) Security features such as JWT-based authentication and RBAC are absent or partial in general-purpose tools. (3) Scalability under high concurrent load is unaddressed in lightweight solutions like Google Forms. (4) Detailed post-exam analytics and report generation are limited in non-commercial platforms. The proposed system directly targets these gaps by offering a dedicated, secure, customizable, and scalable open-source solution.

III. SYSTEM DESIGN AND ARCHITECTURE

The Online Quiz Management System adopts a three-tier architecture: the presentation layer (frontend), application layer (backend), and data layer (database). This layered design enforces separation of concerns, enabling independent development, testing, and scaling of each component. All inter-layer communication is conducted through RESTful APIs using JSON as the data exchange format, ensuring a clean, language-agnostic contract between layers.

A. Architectural Overview

Users interact with the system through the frontend layer (browser-based React.js application). User actions (login, quiz selection, answer submission) generate API requests sent to the backend layer. The backend processes requests, enforces business rules, performs authentication and authorization, and queries the database layer (MySQL). Results are returned as JSON responses and rendered by the frontend. This flow ensures no direct client access to the database, providing a critical security boundary.

B. User Roles and Access Control

Role-Based Access Control (RBAC) governs system access. Two primary roles are defined:

- Admin: Full system authority — creates, edits, activates, and deletes quizzes; manages the question bank; administers user accounts; views all results and generates performance reports; configures system-wide settings.
- Student: Restricted access — self-registers and authenticates; views and attempts available quizzes within configured time limits; views personal score history and feedback; cannot access admin-only endpoints or other students' data.

All API endpoints enforce role verification through JWT token inspection, ensuring students cannot escalate privileges or access administrative functions.

C. Database Schema

The relational schema comprises five normalized entities. User (user_id PK, name, email, password_hash, role, created_at). Quiz (quiz_id PK, title, description, category, total_marks, duration_minutes, created_by FK→User, is_active, created_at). Question (question_id PK, quiz_id FK→Quiz, question_text, option_a, option_b, option_c, option_d, correct_option, marks). Attempt (attempt_id PK, user_id FK→User, quiz_id FK→Quiz, start_time, end_time, score, is_submitted). Result (result_id PK, attempt_id FK→Attempt, total_score, percentage, status). Foreign key constraints maintain referential integrity throughout; cascade deletes ensure consistency when quizzes or users are removed.

D. Data Flow Diagram

At Level 0 (context level), the system has two external entities — Admin and Student — both interacting with the central Online Quiz Management System process. At Level 1, the system decomposes into four sub-processes: User Authentication (validates credentials, issues tokens); Quiz Management (admin CRUD operations on quiz and question data); Quiz Execution (serves questions to students, manages timer state, records responses); and Result Processing (evaluates answers, computes scores, stores and displays results). Each process interacts with corresponding data stores: User DB, Quiz DB, and Result DB.

E. Sequence Diagram Summary

For a student quiz attempt: (1) Student sends login credentials; (2) Backend validates against User DB and returns JWT; (3) Student requests available quizzes; (4) Backend fetches quiz list from Quiz DB; (5) Student selects quiz, backend returns questions; (6) Student submits answers; (7) Backend evaluates against correct answers in Quiz DB; (8) Backend stores result in Result DB; (9) Backend returns score and feedback to student interface.

IV. TECHNOLOGY STACK

A. Frontend (HTML5, CSS3, JavaScript, React.js)

HTML5 provides semantic structure for all page components: registration and login forms, quiz dashboards, question containers, navigation panels, and result displays. Semantic elements (header, main, section, article) improve accessibility and SEO. CSS3 applies responsive styling through Flexbox and CSS Grid layouts, media queries for device adaptation, CSS custom properties for theme consistency, and transition/animation effects for visual feedback. JavaScript adds dynamic client-side behavior —

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

countdown timer logic, answer selection state, question navigation, form validation before API submission, and asynchronous API calls using the Fetch API. React.js manages complex component state through hooks (useState, useEffect, useContext), renders updates through virtual DOM diffing for performance efficiency, and organizes the UI into reusable components (QuizCard, QuestionPanel, TimerDisplay, ResultSummary), simplifying maintenance and extensibility.

B. Backend (Node.js/Express, PHP/Laravel, Django)

Node.js with Express.js serves as the primary backend runtime, offering a non-blocking, event-driven I/O model that efficiently handles hundreds of concurrent API requests without thread contention — critical during simultaneous quiz attempts. Express.js provides lightweight routing, middleware support for authentication, logging, and error handling. PHP with the Laravel framework is used for rapid prototyping and native MySQL integration; Laravel's Eloquent ORM simplifies database interactions, while its built-in CSRF protection and validation layer enhance security. Django (Python) is used for its batteries-included approach: built-in admin panel, ORM, and robust security defaults. All backend implementations expose a uniform RESTful API surface, so the frontend remains implementation-agnostic.

C. Database (MySQL / MongoDB)

MySQL is the primary database, providing ACID-compliant relational data storage with full support for complex JOIN queries needed for report generation (e.g., joining User, Attempt, and Result tables to generate performance summaries). Normalization to Third Normal Form (3NF) eliminates data redundancy. Indexes on user_id, quiz_id, and attempt_id fields reduce query execution time significantly. MongoDB serves as a supplementary store for logging and analytics data where schema flexibility and horizontal scalability are prioritized over strict relational structure.

D. Security Stack

Passwords are hashed using bcrypt with a configurable cost factor (default: 12 rounds), ensuring stored credentials are computationally infeasible to reverse-engineer. JSON Web Tokens (JWT) are issued on successful authentication, signed with a server-side secret key (HS256 algorithm), and required on all protected API endpoints — eliminating stateful session management and enabling horizontal scaling. HTTPS (TLS 1.2+) encrypts all data in transit. Parameterized SQL queries (prepared statements) prevent SQL injection. Output encoding and Content Security Policy headers prevent XSS. SameSite cookie attributes and CSRF tokens mitigate cross-site request forgery.

E. Development Tools

Visual Studio Code served as the primary IDE, with ESLint and Prettier enforcing code style consistency. Git provided distributed version control with a feature-branch workflow: new features developed in isolated branches, merged to main after peer review via GitHub pull requests. XAMPP (Apache + MySQL + PHP) provided the local development stack. Postman was used for API endpoint testing and documentation. GitHub Actions automated linting and unit test execution on every push.

V. IMPLEMENTATION

A. Project Setup and Structure

The project is organized into three top-level directories: /frontend (React application), /backend (Node.js/Express API server), and /database (SQL schema files and seed data). Environment variables (database credentials, JWT secret, server port) are managed through .env files, excluded from version control via .gitignore. The database is initialized by executing the provided schema.sql file, which creates all tables, indexes, and foreign key constraints.

B. User Registration and Login

The registration endpoint accepts POST requests with name, email, username, password, and role fields. Server-side validation checks for required fields, valid email format, minimum password length (8 characters), and uniqueness of email and username in the User table. If validation passes, bcrypt hashes the password and the user record is inserted. The login endpoint validates credentials; on success, a JWT containing user_id and role is signed and returned. Subsequent requests include this token in the Authorization header (Bearer scheme) for verification by the auth middleware.

C. Quiz Creation Module

The admin accesses a multi-step quiz creation wizard through the dashboard. Step 1 captures quiz metadata: title, category, description, total marks, duration, and active status. Step 2 provides a question editor where admins add questions one at a time — entering question text, four option fields, the correct option identifier, and

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

per-question marks. Added questions appear in a reviewable list with edit and delete controls. On final submission, the backend inserts one Quiz record and one Question record per question within a database transaction, ensuring atomicity — either all records are saved or none are.

D. Quiz Attempt Module

The student dashboard displays a quiz catalog with title, category, duration, total marks, and attempt status. Selecting a quiz loads an instruction modal; confirming starts the attempt and records the start_time in the Attempt table. The quiz interface renders one question at a time with radio-button MCQ options. A navigation sidebar shows all question numbers color-coded by status: unattempted (gray), attempted (green), flagged for review (yellow). Student responses are held in React component state during the attempt and only transmitted to the backend as a batch on submission, preventing intermediate partial saves.

E. Timer and Auto Submission

The timer component initializes from the quiz's duration_minutes field (converted to seconds). A setInterval callback decrements the counter every 1000 ms and updates the display. Warning indicators appear at 5 minutes and 1 minute remaining. At zero, clearInterval stops the countdown and the submitQuiz() function fires automatically, transmitting the current answer state to the backend regardless of completion. The backend records end_time, marks the attempt as submitted, and proceeds to evaluation. Server-side validation confirms the submission timestamp is within the allowed window to prevent late submissions manipulated client-side.

F. Result Generation

The evaluation endpoint receives the submitted answer map (question_id → selected_option). It fetches the quiz's correct_option values in a single batch query. Iterating over all questions, it awards marks where student_answer equals correct_option and zero otherwise (negative marking is configurable via quiz settings). Total score, percentage, and pass/fail status (compared against a configurable pass_percentage threshold, default 40%) are computed and stored in the Result table. The response payload includes per-question breakdown (correct vs. incorrect), total score, percentage, and status — displayed immediately in the student result view.

VI. SYSTEM TESTING AND RESULTS

A. Testing Strategy

A comprehensive multi-level testing strategy was employed throughout development. Unit testing validated individual functions and modules in isolation — password hashing, JWT generation/verification, score calculation logic, and timer behavior. Integration testing verified correct data flow across module boundaries: the quiz creation pipeline (admin input → API → database insertion → retrieval by student), the attempt pipeline (question fetch → answer submission → evaluation → result display), and the authentication pipeline (registration → login → protected route access). System testing evaluated the complete application as a unified whole under realistic usage scenarios. User acceptance testing (UAT) involved a group of 10 student volunteers and 2 faculty members who used the system for mock quizzes and provided structured feedback on usability, clarity, and reliability.

B. Functional Test Results

Test#	Test Case	Expected Result	Status
TC-01	Valid user registration	Account created, data saved in DB	Pass
TC-02	Duplicate email registration	Error: email already exists	Pass
TC-03	Login with valid credentials	JWT issued, redirect to dashboard	Pass
TC-04	Login with invalid credentials	Error message, access denied	Pass
TC-05	Admin creates quiz with questions	Quiz stored in DB, visible to students	Pass
TC-06	Student selects and starts quiz	Questions displayed with timer	Pass
TC-07	Auto submission on timeout	Quiz submitted, result generated	Pass
TC-08	Manual submission before timeout	Result calculated and displayed	Pass

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

TC-09	Accurate score calculation	Percentage matches expected value	Pass
TC-10	Unauthorized access to admin endpoint	403 Forbidden returned	Pass

Table II: Functional Test Cases and Results

C. Performance Metrics

Performance testing was conducted using browser developer tools and manual timing under a simulated concurrent load of 10–50 users on a local Apache/Node.js server:

Metric	Observed Value	Remarks
User Login Response	< 1.2 sec	bcrypt adds ~200ms intentionally
Quiz List Load	< 0.8 sec	Indexed DB query
Question Fetch Time	< 1.0 sec	Single JOIN query
Answer Submission	< 0.5 sec	Batch insert + evaluation
Result Display	Instant (< 0.3s)	Server-side evaluation fast
Concurrent Users (local)	Up to 50	No degradation observed
DB Query Response	< 0.4 sec avg	Indexes on FK columns
Timer Accuracy	± 50ms	JavaScript setInterval precision

Table III: System Performance Metrics

D. Security Test Results

Security testing confirmed: (1) SQL injection attempts on login and search inputs returned sanitized errors with no data leakage, as parameterized queries were in effect. (2) XSS payloads submitted in quiz title and question text fields were escaped on output and did not execute in the browser. (3) Direct API calls to admin endpoints using a student JWT returned 403 Forbidden. (4) Expired JWTs were correctly rejected with 401 Unauthorized. (5) HTTPS connection was enforced; plain HTTP requests were redirected. All critical security requirements were satisfied.

E. Usability Feedback

UAT participants rated the system on a 5-point Likert scale. Average scores: Ease of Registration (4.6/5), Quiz Navigation Clarity (4.4/5), Timer Visibility (4.7/5), Result Clarity (4.8/5), Overall Satisfaction (4.5/5). Qualitative feedback highlighted the clean interface and instant result display as standout features. Suggested improvements included mobile-optimized layouts and a review mode showing correct answers after submission — both noted as future enhancements.

VII. DISCUSSION AND LIMITATIONS

A. Analysis of Results

The system achieved all primary objectives: automated quiz management, secure role-based access, timer-enforced examinations, and instant result generation. Functional testing demonstrated 100% pass rate across all ten test cases. Performance metrics confirmed sub-second response times for all core operations under normal load conditions. Security testing validated protection against the most common web application attack vectors (OWASP Top 10). UAT results confirmed strong usability, with average satisfaction scores above 4.4/5 across all categories.

The use of JWT-based stateless authentication proved advantageous for scalability, as the server does not need to maintain session state. The normalized database schema effectively prevented data redundancy and supported efficient querying. The React.js component architecture enabled rapid UI development and consistent state management throughout the quiz attempt workflow.

B. Challenges Encountered

Several technical challenges were addressed during development. Synchronizing the client-side JavaScript timer with server-side submission validation required careful handling of network latency — the server-side timestamp check provides the authoritative boundary. Managing React component state during quiz navigation (ensuring previously selected answers were preserved when navigating backward) required lifting state to a parent component. Handling database transactions atomically during quiz creation (ensuring all questions are saved or none are, if an error occurs mid-save) required explicit transaction management in the backend.

C. Limitations

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

The system has several acknowledged limitations. It supports only MCQ-format questions and cannot evaluate subjective, essay-type, or code-submission responses. Performance testing was conducted on a local server; behavior under production-scale load (hundreds of simultaneous users) requires cloud infrastructure, load balancers, and database connection pooling not implemented in the current version. Security features, while comprehensive for standard threats, lack advanced proctoring capabilities such as eye-tracking, face recognition, or browser lockdown. The system currently requires a stable internet connection with no offline mode. No dedicated mobile application exists; the web interface is responsive but not optimized for touch interaction.

VIII. CONCLUSION & FUTURE WORK

A. Conclusion

The Online Quiz Management System was successfully designed, implemented, and validated as a reliable, efficient, and user-friendly web-based platform for digital examinations. The system fulfills all defined objectives: it provides secure role-based authentication for administrators and students; enables administrators to create and manage quizzes with timer constraints; allows students to attempt quizzes from any internet-connected device and receive instant, automatically computed results; and maintains all data securely in a relational database. Testing across functional, security, performance, and usability dimensions confirmed the system's readiness for deployment in educational and training environments. It represents a practical and effective alternative to paper-based examination management, reducing administrative overhead, eliminating manual grading, and providing a faster, more transparent assessment experience.

B. Future Enhancements

Several directions are planned for future development. AI/NLP-based evaluation will enable automatic grading of descriptive and short-answer questions, significantly expanding the system's applicability beyond MCQ formats. Advanced proctoring features — including real-time video monitoring, AI-based eye-tracking for gaze detection, and browser lockdown to prevent tab switching — will strengthen examination integrity. Mobile applications for Android and iOS will extend accessibility to smartphone users. Cloud deployment on AWS, GCP, or Azure with auto-scaling, containerization (Docker/Kubernetes), and CDN integration will enable production-scale performance. Additional planned features include: multilingual interface support for regional language accessibility; adaptive quiz difficulty based on student performance history; automated certificate generation and digital badging; integration with popular Learning Management Systems (Moodle, Canvas); real-time performance analytics dashboards for educators; and WebSocket-based notifications for live quiz events.

REFERENCES

[1] Mozilla Developer Network, "HTML Documentation." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML
[2] Mozilla Developer Network, "CSS Documentation." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS
[3] Mozilla Developer Network, "JavaScript Documentation." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[4] PHP Group, "PHP Official Documentation." [Online]. Available: https://www.php.net/docs.php
[5] Oracle Corporation, "MySQL Official Documentation." [Online]. Available: https://dev.mysql.com/doc/
[6] I. Sommerville, Software Engineering, 10th ed. Pearson Education, 2015.
[7] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 7th ed. McGraw-Hill, 2019.
[8] W3Schools, "Web Development Tutorials." [Online]. Available: https://www.w3schools.com
[9] Git Documentation, "Version Control." [Online]. Available: https://git-scm.com/doc
[10] GitHub Inc., "GitHub Documentation." [Online]. Available: https://docs.github.com
[11] ReactJS, "React – A JavaScript Library for Building User Interfaces." [Online]. Available: https://reactjs.org
[12] Node.js Foundation, "Node.js Documentation." [Online]. Available: https://nodejs.org/en/docs
[13] Django Software Foundation, "Django Documentation." [Online]. Available: https://docs.djangoproject.com
[14] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015.
[15] OWASP Foundation, "OWASP Top Ten." [Online]. Available: https://owasp.org/www-project-top-ten/