Volume: 13 Issue: VI Month of publication: June 2025

DOI: https://doi.org/10.22214/ijraset.2025.71322

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VI June 2025- Available at www.ijraset.com

Design and Implementation of Quiz Application using MERN

Abhishek Kushwaha[1], Rajendra Kumar Gupta[2]
Madhav Institute of Technology and Science, Gwalior- 474005

***Abstract:** This paper presents the design and development of Quiz Application built using MERN stack – MongoDB, Express.js, React.js and Node.js. The project gives me hand-on experience in working with full-stack web development project. Quiz Applications has a great value in educational sector, also it promotes digital learning and reduces the manual effort for conducting examination. Traditional way of conducting examinations takes lot of manual efforts such as preparing paper-based question sets, Distributing and collecting answer sheets, manually evaluating responses.The Application allows the Teachers to create and manage quizzes through user-friendly teacher panel and students can attempt quizzes in a student panel. The complete project is divided into three parts i.e. Student Panel, Teacher Panel and Database. The student Panel and Teacher panel both are 3-tier architecture based with shared database. Student panel consist student authentication, quiz attempt, answer submission and get result instantly. Whereas teacher panel consist of quiz creation, quiz management, seeing result as well as adding and managing students.Unlike generic quiz applications that often rely on self-registration and static question banks, the proposed system introduces a controlled environment where teachers manage user credentials, quiz creation, and data resets. Its real-time result evaluation, modular design with separate panels, and centralized database integration offer a robust, scalable solution specifically tailored for academic institutions. This distinct approach enhances usability, security, and administrative control compared to existing solutions*

***Keywords:** Quiz Application, MERN Stack, MongoDB, Express.js, React.js, Node.js, Educational Institutes, Full-Stack Development*

I. INTRODUCTION

The shift towards digital learning has necessitated the development of intelligent and interactive assessment systems. Traditional quizzes lack adaptability and real-time feedback. This work proposes a web-based quiz system using MERN stack technologies, enabling dynamic quiz creation, automatic result evaluation, and secure user authentication tailored for both students and teachers.The advancement of digital education has created a demand for efficient and scalable online quiz platforms. Traditional quiz methods, which rely heavily on paper-based systems, are time-consuming, error-prone, and inefficient for both teachers and students. While numerous quiz platforms exist, many are either overly complex, expensive, or lack customization options for institutional needs.

This research focuses on the development of a quiz application using the MERN stack that caters specifically to educational institutes. The application enables teachers to easily create and manage quizzes, while allowing students to take quizzes securely through personalized logins. Although similar systems exist, this project adds value by offering a self-hosted, open-source solution tailored for specific academic workflows.

II. LITERATURE REVIEW

Several studies and projects have explored the development of online quiz systems and the use of modern web technologies like the MERN stack. These studies emphasize the importance of real-time evaluation, user role management, and data persistence in educational platforms.

In [1], a MERN-based quiz application was implemented to educate users about social media hoaxes. It demonstrated effective use of React for UI rendering and MongoDB for scalable data storage, supporting real-time feedback for quiz attempts.

Similarly, [2] presented a quiz portal designed for educational institutions. The system provided features such as student login, question randomization, and performance evaluation, showcasing the adaptability of MERN for academic purposes.

Another study [3] introduced an e-learning system integrating quizzes as a part of broader learning content. The authors highlighted the importance of intuitive UI design and backend robustness in handling concurrent user sessions, a challenge addressed using Express.js and MongoDB.

Despite the availability of various quiz applications, many lack modularity or role-based features. This paper proposes a solution that distinguishes itself by combining real-time feedback, teacher-controlled quiz generation, and dynamic result storage, all supported by a unified MERN infrastructure

III. SYSTEM ARCHITECTURE

The proposed quiz application follows a modular and scalable architecture, comprising three interconnected components: the Student Panel, the Teacher Panel, and a shared database. Both the Student and Teacher Panels follow a **three-tier architecture**, consisting of the client side (React.js), a backend server (Node.js with Express), and the database (MongoDB). Each component is described below:

A. Student Panel

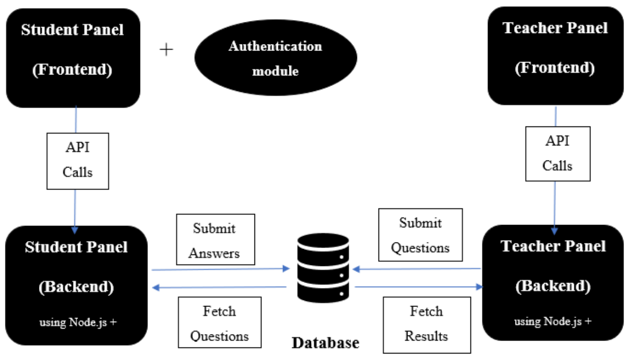
The Student Panel represents the user-facing interface where students can log in, attempt quizzes, and receive real-time results. It interacts with the backend server through RESTful APIs to fetch questions and submit responses. The server processes the data and communicates with MongoDB to validate answers and store results. This panel is built using React.js for a responsive UI, Node.js and Express.js for backend logic, and MongoDB for persistent storage.

B. Teacher Panel

The Teacher Panel is an administrative interface used by teachers or quiz organizers to create quizzes, manage student credentials, and monitor results. It also follows a three-tier web architecture and interacts with the same backend and database as the Student Panel. The separation of the Teacher Panel ensures a secure and organized environment for content and user management.

C. Shared MongoDB Database

A centralized MongoDB database serves as the persistent storage layer for both panels. It stores quiz questions, student login credentials, and attempt results. Using a shared database ensures data consistency, simplifies synchronization between modules, and supports efficient access control.



IV. FUNCTIONAL MODULES

The application consists of two primary modules: the Teacher Panel and the Student Panel. Both modules follow a three-tier architecture comprising a frontend, a backend, and a database. They share a common database to ensure data consistency and ease of management.

The Student Panel enables users to attempt quizzes and view their results. The frontend, developed using React.js, fetches quiz data from the backend and presents it in a user-friendly format. Students can select answers and submit the quiz within a predefined time limit. Upon submission, the results are calculated and displayed immediately. The backend is implemented using Node.js and Express.js, which handle all server-side operations. It manages communication between the frontend and the MongoDB database, compares student responses with the correct answers, and calculates the final score. The backend also includes API routes for fetching questions from the database, authenticating students, and posting the result data to the database. The shared MongoDB database stores student attempts and results, utilizes the same quiz data created by teachers, and ensures that all records remain synchronized across both panels.

The Teacher Panel is responsible for creating and managing quizzes. Its frontend is built with React.js to offer a dynamic and user-friendly interface. Teachers can input the number of questions for each quiz, enter each question along with multiple-choice options, correct answers, marks per question, and the passing criteria.

Additionally, they can add student credentials such as enrollment numbers and passwords for those who are expected to attempt the quiz. Teachers are also provided with the functionality to delete existing questions or remove result data from the database when necessary. They can view the results of students who have completed the quiz, including their scores and pass/fail status.

The backend, also developed using Node.js and Express.js, handles server-side tasks such as receiving quiz data from the frontend and storing it in the MongoDB database in a structured JSON format. It includes API routes for saving new quizzes, adding student credentials, deleting specific questions or result data, and retrieving student results for display on the frontend. The shared MongoDB database stores quiz questions and their corresponding answers, along with student login credentials. Each quiz entry is timestamped using a "createdAt" field to facilitate proper record-keeping. The quiz structure is maintained in a JSON-like format for consistency and efficient data retrieval.

V. IMPLEMENTATION

The implementation of this Quiz Application is based on a three-tier architecture comprising the Student Panel, Teacher Panel, and a centralized shared MongoDB database. Each module has its own frontend and backend layers, communicating with the database through structured API routes.

The Student Panel acts as the user-facing interface where students can log in, attempt quizzes, and view their results. Built using React.js, this panel provides a responsive UI and connects to the backend via RESTful APIs built with Node.js and Express.js. The backend processes data, validates answers, and communicates with MongoDB to fetch and store quiz results. The student module also uses Redux for efficient state management, with custom reducer functions to track user login status, question data, and scores. Helper functions such as checkUserExist() and earnPointNumber() are implemented to manage logic like user authentication and point calculation, enhancing user interaction. The student panel ensures a smooth and secure quiz-taking experience, while maintaining modularity through components and routes like /quiz, /result, and /login.

The Teacher Panel is designed for administrative use, enabling teachers to create quizzes, manage student credentials, view and delete results, and configure quiz details such as question count, multiple-choice options, correct answers, marks per question, and the passing criteria. This panel is also built using React.js, employing the same Redux-based state management, along with Axios for sending HTTP requests to the backend. The backend of the teacher panel, using Express.js and Node.js, handles server-side operations such as receiving new quiz data and student credentials, storing them in MongoDB, and retrieving them for result management. API routes such as /api/questions, /api/result, /api/login, and /api/students enable CRUD operations. Teachers can view student performance in real time and have access to quiz metadata, including time stamps and pass/fail status.

The shared MongoDB database serves as the centralized storage unit for both panels, maintaining collections for quizzes, student credentials, answers, and results. It ensures synchronization of data across both modules, enabling consistent access and update operations. Quiz data is stored in structured JSON format, with each quiz entry time-stamped using a createdAt field to support efficient tracking and retrieval. Mongoose is used to define schemas and facilitate interactions between the server and the database. By centralizing data storage and separating the student and teacher functionalities into two panels, the application offers secure user access, simplified data management, and a highly maintainable structure.

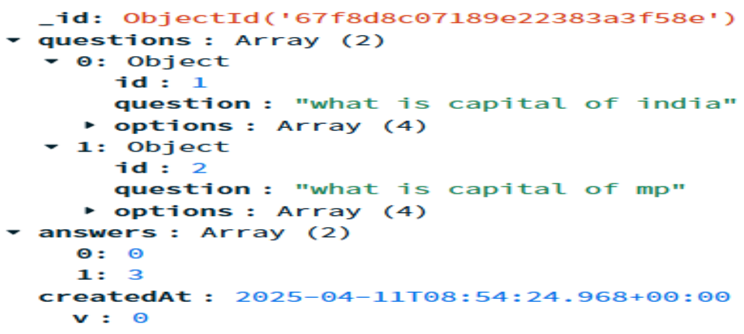
A. Database Design

Used MongoDB Atlas to store the data for the application. The database consist of three different collection –

1) Questions Collection

Purpose:Stores quiz questions along with options and their correct answers.

Structure Example:

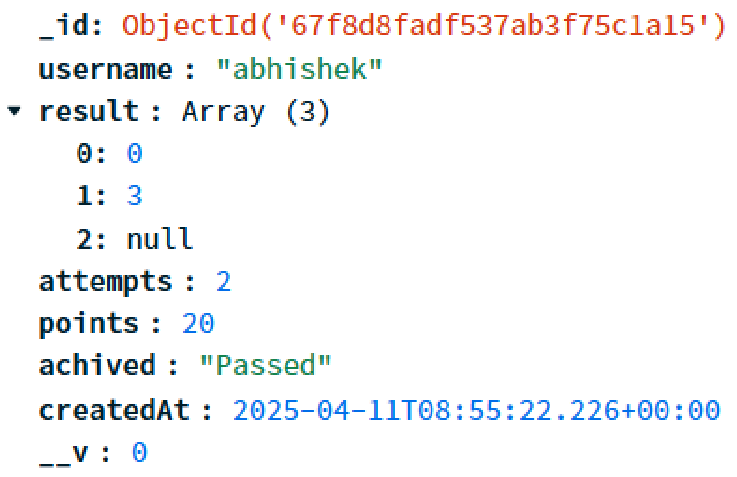


```
    _id: ObjectId('67f8d8c07189e22383a3f58e')
  ▾ questions : Array (2)
    ▾ 0: Object
        id : 1
        question : "what is capital of india"
      ▸ options : Array (4)
    ▾ 1: Object
        id : 2
        question : "what is capital of mp"
      ▸ options : Array (4)
  ▾ answers : Array (2)
      0: 0
      1: 3
    createdAt : 2025-04-11T08:54:24.968+00:00
      v : 0
```

2) Results Collection

Purpose:Stores the quiz results of students i.e. their number of questions attempted, their marks, and whether they passed or failed the quiz , including scores and timestamps.

Structure Example:

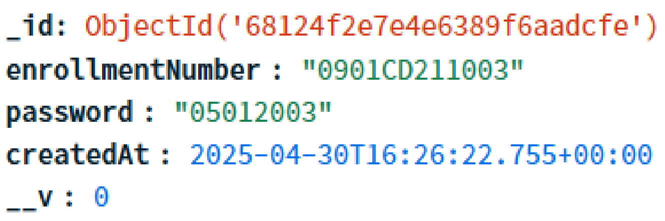


```
  _id: ObjectId('67f8d8fadf537ab3f75c1a15')
  username : "abhishek"
▾ result : Array (3)
    0: 0
    1: 3
    2: null
  attempts : 2
  points : 20
  achived : "Passed"
  createdAt : 2025-04-11T08:55:22.226+00:00
  __v : 0
```

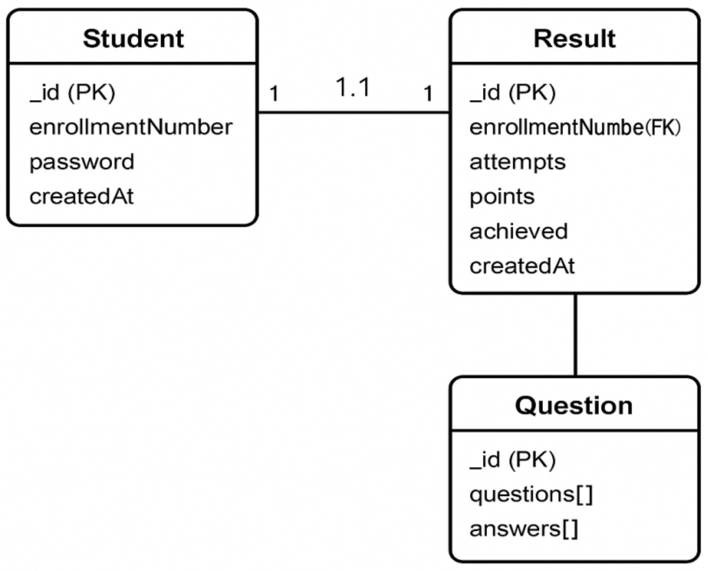
3) Students Collection

Purpose:Stores registered student information and credentials like enrollment Number and password.

Structure Example:



```
_id: ObjectId('68124f2e7e4e6389f6aadcfe')
enrollmentNumber : "0901CD211003"
password : "05012003"
createdAt : 2025-04-30T16:26:22.755+00:00
__v : 0
```



B. Data flow

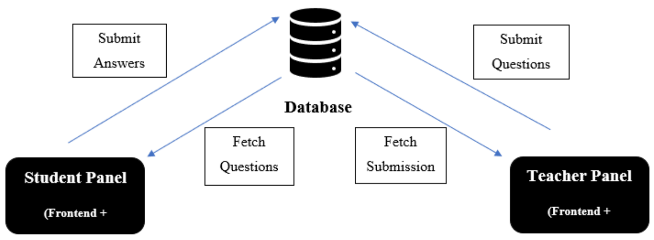
The data flow diagram illustrates the interaction between the three core components of the Quiz Application: the Student Panel, the Teacher Panel, and the centralized Database. This architecture follows a three-tier design, where both panels interact with the database via backend services to maintain modularity and scalability.

The Student Panel allows users to fetch quizzes and submit their answers. Upon login, students retrieve quiz questions from the database through the backend API. After completing the quiz, their responses are submitted back to the database for storage and evaluation. These operations typically utilize HTTP GET requests for fetching data and POST requests for submitting answers.

The Teacher Panel is responsible for quiz creation and result monitoring. Teachers can define a set of questions for each quiz, specify options, correct answers, assign marks per question, and set the passing criteria. Once a quiz is created, the data is submitted to the database, also using POST requests. Teachers can also fetch student submissions and view evaluation results using GET requests.

At the center of this structure is a centralized Database, which stores all quiz-related information, including questions, answers, student submissions, and user credentials. The database ensures data consistency and synchronization between the two panels. All data interactions go through the backend, built using Node.js and Express.js, which handles request routing, validation, and response generation.

This modular separation enhances maintainability and ensures secure data handling, while the use of MongoDB enables scalable and efficient storage of semi-structured quiz data. The diagram effectively captures the system's flow of information, highlighting the key data transactions that support quiz management and evaluation processes.



VI. CHALLENGES AND SOLUTIONS

During the development of the Quiz Application, several technical challenges were encountered, each offering valuable learning experiences. One notable issue arose while implementing the quiz options, where the radio button selection consistently defaulted to the first option, regardless of user interaction. After thorough debugging, it was discovered that the problem stemmed from incorrect usage of quotation marks in JSX. Specifically, apostrophes (') were used instead of backticks (``) when dynamically assigning the value or name attributes to radio buttons. Replacing them with the correct syntax resolved the issue and reinforced the critical importance of attention to detail in JSX syntax.

Another significant challenge involved the behavior of the insertMany() method in MongoDB while submitting questions from the Teacher Panel to the database. Initially perceived as a coding error, further investigation revealed that the issue was due to changes introduced in recent versions of MongoDB. The method's behavior had been updated, necessitating minor modifications in the codebase to comply with the revised usage. This experience underscored the importance of regularly consulting official documentation and staying informed about updates in libraries and databases used in development.

VII.FUTURE SCOPE

The current implementation of the Quiz Application provides a strong foundation; however, there are several areas that can be enhanced in future iterations to improve scalability and functionality. One key enhancement is the implementation of a teacher registration system within the Teacher Panel. This would enable individual teachers to securely register, manage their own quizzes, and access personalized dashboards, thereby offering greater flexibility and multi-user support.

Additionally, the existing system is limited to supporting only one active quiz at a time. In future versions, this limitation can be addressed by introducing functionality that allows the creation and management of multiple quizzes concurrently. This would be particularly beneficial for institutions conducting assessments for different subjects or batches simultaneously.
Another important feature that can be integrated is quiz time specification. This would allow teachers to define a time window during which the quiz remains open, automatically restricting access once the allotted time has elapsed. Incorporating such time-based access control would enhance the practicality of the application for real-time examinations and scheduled assessments.

VIII. CONCLUSION

The development of the Quiz Application using MERN provided hands-on experience in building a full-stack web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The system successfully meets the goal of allowing teachers to create quizzes and students to attempt them in a secure, timed, and structured environment.Throughout the development process, various essential concepts such as React routing, Redux state management, API communication, and MongoDB data modelling were implemented. The challenges faced—ranging from frontend quirks to backend data handling—helped solidify foundational knowledge and enhance problem-solving skills.This project not only served as an academic requirement but also as a practical application of modern web development technologies. The final product is scalable and can be enhanced in the future with features like admin dashboard, analytics, authentication, and email notifications

REFERENCES

[1] Sharma, A., & Patel, R. (2021). Design and Implementation of Online Quiz Application Using MERN Stack. International Journal of Computer Applications, 183(9), 24–29.
[2] MongoDB Documentation. (2024). insertMany() Behavior and Options. Available at: https://www.mongodb.com/docs/manual/ reference/ method/db.collection.insertMany/
[3] ReactJS Documentation. (2024). JSX Syntax and Best Practices. Available at:https://reactjs.org/docs/introducing-jsx.html
[4] Node.js Documentation. (2024). Building Scalable APIs with Express. Available at: https://nodejs.org/en/docs/
[5] Mozilla Developer Network (MDN). (2024). Working with Forms in HTML and JavaScript. Available at: https://developer.mozilla.org/