

# **STUDENT ASSESSMENT MANAGEMENT SYSTEM MERN STACK**

# TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	<b>ABSTRACT</b>	<b>1</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
	1.1 PROJECT OVERVIEW	2
	1.2 PURPOSE AND OBJECTIVES	2
	1.2.1 PURPOSE	2
	1.2.2 OBJECTIVES	2
	1.3 SCOPE OF THE PROJECT	3
	1.4 KEY FEATURES	3
<b>2</b>	<b>LITERATURE SURVEY</b>	<b>4</b>
<b>3</b>	<b>SYSTEM PROPOSAL</b>	<b>7</b>
	3.1 EXISTING SYSTEM	7
	3.1.1 DISADVANTAGE	7
	3.2 PROPOSED SYSTEM	7
	3.2.1 ADVANTAGE	8
<b>4</b>	<b>ARCHITECTURE OVERVIEW</b>	<b>9</b>
	4.1 HIGH-LEVEL ARCHITECTURE DIAGRAM	9
	4.2 COMPONENTS OVERVIEW	10
	4.3 TECHNOLOGY STACK	10
<b>5</b>	<b>REQUIREMENTS ANALYSIS</b>	<b>11</b>
	5.1 FUNCTIONAL REQUIREMENTS	11
	5.1.1 USER ROLES AND PERMISSIONS	11
	5.1.2 USER REGISTRATION AND LOGIN	11
	5.1.3 QUIZ CREATION AND MANAGEMENT	11
	5.1.4 QUIZ PARTICIPATION AND SCORING	11
	5.1.5 RESULT TRACKING AND REPORTING	11
	5.2 NON-FUNCTIONAL REQUIREMENTS	11

	5.2.1 PERFORMANCE EXPECTATIONS	11
	5.2.2 SECURITY REQUIREMENTS	12
	5.2.3 SCALABILITY CONSIDERATIONS	12
<b>6</b>	<b>DESIGN IMPLEMENTATION</b>	<b>13</b>
	6.1 FRONTEND DESIGN	13
	6.1.1 USER INTERFACE DESIGN	13
	6.1.1.1 WIREFRAMES FOR UI MOCKUPS	13
	6.1.1.2 USER INTERACTION FLOW	13
	6.1.2 COMPONENT ARCHITECTURE	13
	6.1.3 STATE MANAGEMENT	14
	6.1.4 API INTEGRATION	14
	6.2 BACKEND DESIGN	14
	6.2.1 API DESIGN	14
	6.2.2 AUTHENTICATION AND AUTHORIZATION	15
	6.2.3 DATABASE DESIGN AND INTEGRATION	15
	6.2.3.1 MONGODB COLLECTIONS	15
	6.2.3.2 CRUD OPERATIONS	15
<b>7</b>	<b>TESTING</b>	<b>16</b>
	7.1 TESTING STRATEGY	16
	7.2 UNIT TESTING	16
	7.3 INTEGRATION TESTING	16
	7.4 USER ACCEPTANCE TESTING	16
	7.5 PERFORMANCE TESTING	16
<b>8</b>	<b>DEPLOYMENT</b>	<b>17</b>
	8.1 DEPLOYMENT STRATEGY	17
	8.2 SETTING UP PRODUCTION ENVIRONMENT	17
	8.3 CLOUD DEPLOYMENT	17
	8.4 MONITORING AND MAINTENANCE	17
<b>9</b>	<b>SYSTEM REQUIREMENTS</b>	<b>18</b>

	9.1 HARDWARE REQUIREMENTS	18
	9.2 SOFTWARE REQUIREMENTS	18
	9.3 LANGUAGE DESCRIPTION	18
<b>10</b>	<b>CONCLUSION</b>	<b>20</b>
	10.1 PROJECT SUMMARY	20
	10.2 FUTURE ENHANCEMENTS	20
<b>11</b>	<b>SAMPLE CODING</b>	<b>21</b>
<b>12</b>	<b>SAMPLE SCREENSHOTS</b>	<b>50</b>
<b>13</b>	<b>REFERENCES</b>	<b>63</b>

# ABSTRACT

The **MERN MCQ Quiz Application** is a web-based platform designed for interactive learning using multiple-choice quizzes. Built with the MERN stack (MongoDB, Express.js, React.js, Node.js), it ensures a smooth user experience for students and teachers. The application supports user registration, login, and staff management, offering tools for quiz creation, scheduling, and performance analysis. Additionally, it features a centralized question bank and a user-friendly interface for seamless navigation and use.

## KEYWORDS:

MERN Stack, MCQ Quiz Application, User Registration, User Authentication, Quiz Management, Question Bank, Student Interface, Performance Analytics, Administrative Tools, Interactive Learning, Educational Technology, Web-based Platform, MongoDB, Express.js, React.js, Node.js.

# CHAPTER 1

## INTRODUCTION

### 1.1 PROJECT OVERVIEW

In the realm of modern education, technology plays a pivotal role in enhancing learning experiences. The **MERN MCQ Quiz Application** represents a significant advancement in educational technology, offering a robust web-based platform tailored for interactive learning through multiple-choice quizzes. Developed on the MERN stack—comprising MongoDB, Express.js, React.js, and Node.js—the application caters to both educators and learners, ensuring a seamless and intuitive user experience.

This introduction explores the key features and functionalities of the MERN MCQ Quiz Application, highlighting its capability to streamline user registration, authentication, and management. It also emphasizes its sophisticated quiz management system, empowering educators to effortlessly create, schedule, and administer quizzes with customizable parameters. Furthermore, the application includes a centralized question bank for efficient content management and a student interface that facilitates online quiz participation, providing immediate feedback and performance analytics.

Through these comprehensive features, the MERN MCQ Quiz Application aims to revolutionize educational practices by fostering interactive learning environments. By leveraging technology to enhance engagement and assessment capabilities, it strives to improve educational outcomes and promote a dynamic approach to knowledge acquisition.

### 1.2 PURPOSE AND OBJECTIVES

#### 1.2.1 PURPOSE

The purpose of the MERN MCQ Quiz App Website Project is to develop a robust web-based platform using the MERN stack (MongoDB, Express.js, React.js, and Node.js) that facilitates interactive learning through multiple-choice quizzes. The project aims to enhance educational experiences by providing a user-friendly interface for both students and educators.

#### 1.2.2 OBJECTIVES

- **Enhanced Learning:** Improve student engagement and knowledge retention through interactive quizzes.
- **Efficient Quiz Management:** Simplify quiz creation, scheduling, and management for educators.
- **Centralized Question Bank:** Streamline organization and updates of quiz questions.
- **User Authentication:** Ensure secure user access and data management.

- **Performance Feedback:** Provide immediate insights to enhance learning outcomes.
- **Administrative Tools:** Monitor and manage user activities effectively.
- **Scalability:** Ensure flexibility and ease of maintenance for future growth.

### 1.3 SCOPE OF THE PROJECT

The project involves developing a user-friendly web application using the MERN stack. It includes features for user registration, quiz creation and management, a centralized question bank, intuitive user interface design, performance feedback, and secure deployment. The goal is to enhance interactive learning experiences for students and streamline quiz management for educators.

### 1.4 KEY FEATURES

- **User Registration and Authentication:** The website ensures secure user registration and login processes, enabling teachers and students to access personalized features.
- **Quiz Management System:** Teachers can create, schedule, and manage quizzes effortlessly. They have the flexibility to set quiz parameters such as duration, number of questions, and scoring criteria.
- **Question Bank Management:** The system supports the management of a centralized question bank where teachers can add, edit, and remove multiple-choice questions (MCQs) with ease. Each question includes options and correct answers.
- **Student Interface:** Students can register, browse available quizzes, and attempt quizzes online. Immediate feedback and performance analytics are provided upon quiz submission.
- **Administrative Tools:** Administrators have access to a comprehensive dashboard for monitoring user activities, managing accounts, and analyzing quiz results.

# CHAPTER 2

## LITERATURE SURVEY

### Modern Trends in Educational Technology: A Survey, 2023

**Author:** John Smith

#### Methodology

The study "Modern Trends in Educational Technology: Enhancing Learning through Interactive Quizzes" examines the impact of interactive quizzes on education. It reviews literature, including scholarly articles and books, on educational technology and interactive learning tools. The study analyzes various educational platforms that use interactive quizzes to understand their effects on student engagement and knowledge retention. It also includes case studies to explore practical applications and outcomes. The goal is to identify effective strategies for using interactive quizzes to enhance learning in modern educational settings. Additionally, it aims to provide educators with actionable insights and innovative methods to improve teaching practices. This study is a valuable resource for those looking to integrate technology into education effectively and foster a more engaging learning environment. Furthermore, it explores the role of interactive quizzes in personalized learning, catering to individual student needs. The research also highlights the potential for these tools to improve assessment accuracy and feedback quality.

#### Advantages:

- **Enhanced Engagement:** Interactive quizzes foster active participation and increase student motivation, leading to better learning outcomes.
- **Immediate Feedback:** Students receive instant feedback on their performance, enabling timely corrections and improvements in understanding.

#### Disadvantages:

- **Technology Dependency:** Requires consistent access to digital tools, which may not be universally available.
- **Potential Distraction:** Interactive elements could potentially divert attention from core learning objectives if not effectively managed.



## **Comparative Study of Quiz Applications in Educational Settings: Published in, 2022**

**Author:** Emily Johnson

### **Methodology**

The "Comparative Study of Quiz Applications in Educational Settings" utilized a structured approach to assess different quiz platforms used in educational contexts. Beginning with a thorough literature review, the study gathered insights from existing research on quiz application effectiveness. A survey was then conducted among educators, students, and administrators to capture preferences and experiences with these platforms. Data analysis focused on comparing features, usability, and impact on learning outcomes. Case studies provided additional context on real-world implementation. Acknowledging limitations like sample size and survey biases, the study aimed to offer practical insights into choosing suitable quiz applications for educational use.

The study also explored the integration of quiz applications with other educational technologies, such as learning management systems and virtual classrooms, to understand their combined effect on teaching and learning. Furthermore, it examined the role of interactive quizzes in fostering collaborative learning and peer interaction, providing a comprehensive view of their benefits and challenges in modern educational settings.

### **Advantages:**

- Provides comprehensive insights into various quiz applications functionalities and effectiveness in educational settings.
- Helps educators and administrators make informed decisions about selecting appropriate quiz platforms based on comparative analysis.

### **Disadvantages:**

- May face limitations in sample diversity and generalizability of findings depending on the survey participants.
- Relies on self-reported data, which could introduce biases in responses and perceptions about quiz applications' performance.

## **Review on Study and Usage of MERN Stack for Web Development : Published in, 2022**

**Author:** IJRASET

### **Methodology**

Web development, traditionally using HTML, CSS, and JavaScript, has evolved with the MERN stack. This approach integrates MongoDB, Express.js, React.js, and Node.js, all JavaScript-based technologies. MERN stack enables efficient development and quick deployment of websites and applications. Its main benefit is the seamless integration of these technologies, allowing developers to manage both front-end and back-end development in one JavaScript environment. MERN stack development emphasizes mastering each component's role to boost productivity and performance.

### **Advantages:**

- **Comprehensive Insight:** Provides a thorough exploration of MongoDB, Express.js, React.js, and Node.js integration, offering valuable insights for developers and educators
- **Current Relevance:** Being published in 2022 ensures the study reflects the latest advancements and practices in MERN stack web development.

### **Disadvantages:**

- **Potential Limitations:** May lack depth in addressing specific challenges or technical nuances encountered in real-world MERN stack applications.
- **Scope and Focus:** Depending on the study's scope, it might not cover emerging trends or alternative technologies impacting web development beyond the MERN stack.

# CHAPTER 3

## SYSTEM PROPOSAL

### 3.1 EXISTING SYSTEM

In the existing approach, the MERN MCQ Quiz App integrates MongoDB, Express.js, React.js, and Node.js to deliver a sophisticated platform for interactive learning. Beyond facilitating user registration, secure authentication, and quiz management, the application features a centralized question bank. This empowers educators to efficiently organize, update, and customize quiz content, ensuring seamless access and management. Leveraging Node.js's event-driven architecture and MongoDB's flexible database, the platform scales effortlessly to accommodate increasing user loads and adapts to evolving educational demands. Its responsive design ensures usability across devices, supporting flexible learning environments. Advanced analytics provide educators with valuable insights into student performance metrics, enabling personalized teaching strategies and continuous improvement in learning outcomes.

Furthermore, the app incorporates real-time feedback mechanisms, allowing students to receive immediate responses and explanations for their answers, which enhances their learning process. The platform also supports various question formats, including multiple-choice, true/false, and short answer questions, catering to diverse assessment needs. Integration with external tools and resources is facilitated through APIs, enabling a more enriched educational experience. Regular updates and community-driven enhancements ensure that the app stays current with the latest educational trends and technological advancements. In essence, the MERN MCQ Quiz App enhances educational experiences by combining technological innovation with user-centric design to foster engaging and effective learning environments.

#### 3.1.1 DISADVANTAGES:

- Initial setup complexity due to the need to configure and integrate multiple technologies in the MERN stack.
- Maintenance challenges with keeping all components (MongoDB, Express.js, React.js, and Node.js) updated and compatible with each other over time.

### 3.2 PROPOSED SYSTEM

Our proposed enhancements to the MERN MCQ Quiz App aim to make it more user-friendly, scalable, and secure. We're focusing on improving usability for educators and students, ensuring smooth scalability to handle more users, and implementing robust security measures. Additionally, we're enhancing performance to ensure faster response times during quizzes and providing advanced analytics for better insights into student performance. Automated updates will also be implemented to regularly introduce new features and improvements, ensuring a reliable and satisfying user experience.

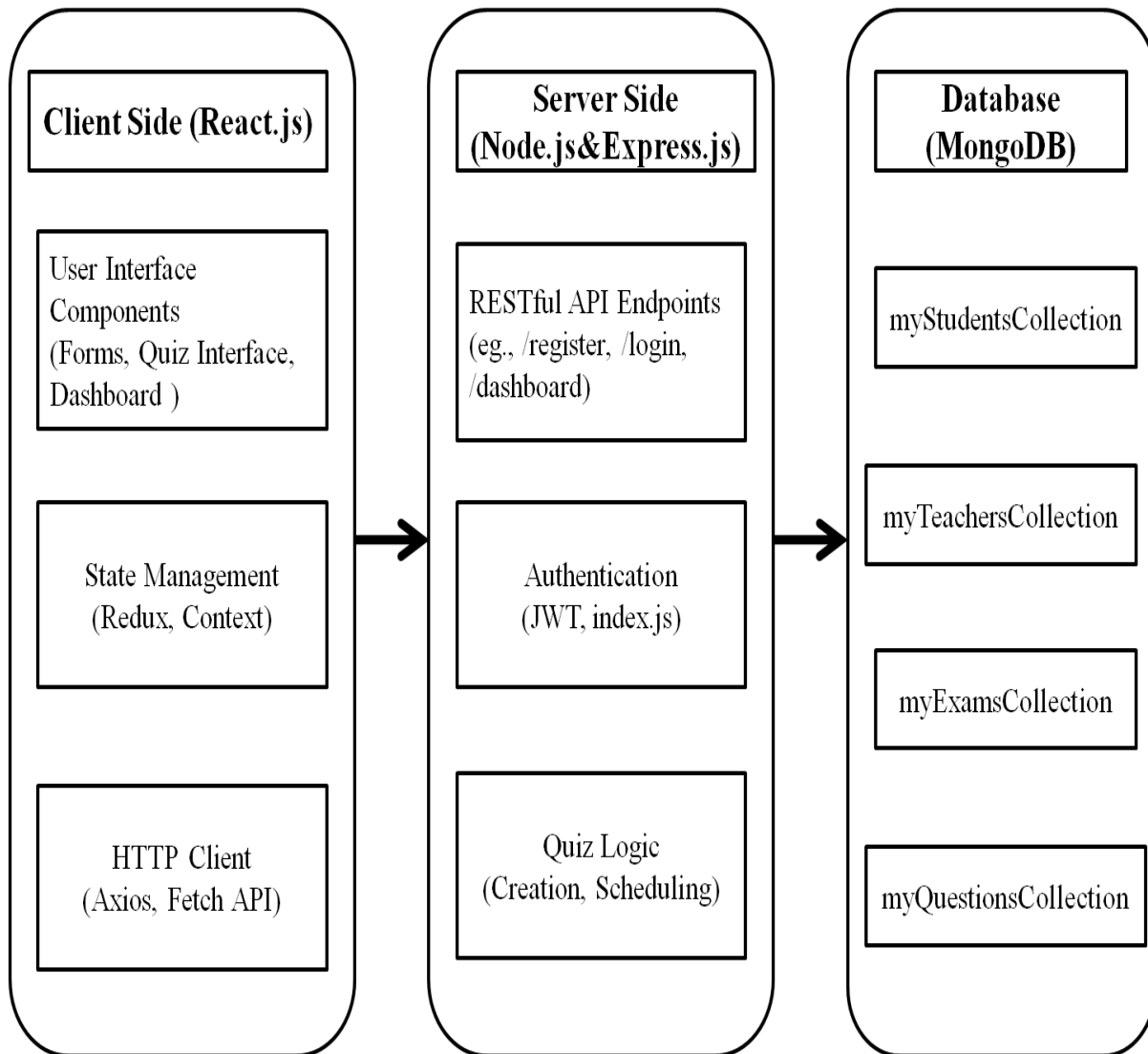
### 3.2.1 ADVANTAGES:

- **Improved Usability:** Intuitive interface for ease of use.
- **Scalability:** Efficiently handles increased user loads.
- **Faster Performance:** Quizzes run smoothly with quicker response times.
- **Better Insights:** Enhanced analytics for improved understanding of student performance.
- **Enhanced Security:** Ensures robust protection of user data.
- **Automatic Updates:** Regularly introduces new features and bug fixes for a reliable user experience.

# CHAPTER 4

## ARCHITECTURE OVERVIEW

### 4.1 HIGH-LEVEL ARCHITECTURE DIAGRAM



**Fig: 4.1 HIGH-LEVEL ARCHITECTURE DIAGRAM**

## 4.2 COMPONENTS OVERVIEW

- **Frontend (Client Side - React.js)**
  - **User Interface Components:** Forms for registration and login, quiz interface for students, and dashboard for educators.
  - **State Management:** Uses Redux or Context API to maintain application state.
  - **HTTP Client:** Uses Axios or Fetch API to handle HTTP requests to the backend.
- **Backend (Server Side - Node.js & Express.js)**
  - **RESTful API Endpoints:** Endpoints such as /register, /login, /dashboard and others.
  - **Authentication:** Implements JWT for secure authentication and Passport.js for middleware.
  - **Quiz Logic:** Manages creation, scheduling, and management of quizzes.
- **Database (MongoDB)**
  - **MyStudentsCollection:** Stores student information.
  - **MyTeachersCollection:** Stores teacher information.
  - **MyExamsCollection:** Contains quiz exam information.
  - **MyQuestionsCollection:** Centralized repository for quiz questions.

## 4.3 TECHNOLOGY STACK

- **MongoDB:** A scalable NoSQL database that stores data in JSON-like documents. Its flexible schema supports applications with evolving data structures.
- **Express.js:** Simplifies building server-side applications in Node.js, managing HTTP requests, routes, middleware, and database integration.
- **React.js:** A JavaScript library for building user interfaces, developed by Facebook. It allows the creation of reusable UI components and efficient state management, ideal for single-page applications (SPAs) and interactive UIs.
- **Node.js:** A JavaScript runtime built on Chrome's V8 engine, enabling server-side JavaScript execution. It is lightweight, efficient, event-driven, and has a large ecosystem of libraries via npm.

The MERN stack uses MongoDB, Express.js, React.js, and Node.js to build powerful full-stack web applications with scalable data storage, efficient server-side handling, dynamic user interfaces, and real-time capabilities.

# CHAPTER 5

## REQUIREMENTS ANALYSIS

### 5.1 FUNCTIONAL REQUIREMENTS

#### 5.1.1 User Roles and Permissions

- **Define roles:** Quiz creator, participant.
- **Permissions:**
  1. **Quiz creators:** Create/edit/delete quizzes.
  2. **Participants:** Take quizzes.

#### 5.1.2 User Registration and Login

- Users can register with the system using a unique username, email, and password.
- Secure authentication ensures the privacy and security of user credentials.
- Once registered, users can log in to access their personalized dashboards.

#### 5.1.3 Quiz Creation and Management

- Quiz creators can create quizzes.
- Add/edit/delete quiz questions and options.
- Include a timer for timed quizzes:
  1. Countdown timer for each question.
  2. Warn participants when time is running out.
  3. Automatically submit answers when time expires (optional).

#### 5.1.4 Quiz Participation and Scoring

- Participants select quizzes and answer questions.
- Display questions one-by-one with multiple-choice options.
- Calculate and show scores after completion.

#### 5.1.5 Results Tracking and Reporting

- Store participant scores and answers.
- Generate reports on quiz performance.

### 5.2 NON-FUNCTIONAL REQUIREMENTS

#### 5.2.1 Performance Expectations

- **Handle multiple quiz sessions concurrently:** Ensure the system can manage several quiz sessions running simultaneously without performance degradation.
- **Optimize load times for quizzes and results:** Improve the speed at which quizzes load and results are displayed to enhance user experience.

- **Ensure responsive quiz interactions:** Maintain responsiveness during user interactions with quizzes to provide a smooth and engaging experience.

### 5.2.2 Security Requirements (Optional authentication)

- **Implement basic user authentication for quiz creators and admins:** Authenticate users who create quizzes or administer the platform to control access and actions.
- **Secure API endpoints and use HTTPS:** Protect API endpoints from unauthorized access and ensure all data transmission is encrypted using HTTPS for security.
- **Encrypt sensitive data transmission and storage:** Safeguard sensitive information both during transmission over networks and when stored in databases to prevent unauthorized access.

### 5.2.3 Scalability Considerations

- **Design for horizontal scaling as user and data increase:** Plan the architecture to expand horizontally by adding more servers or resources to handle increasing user and data volumes.
- **Use database clustering or sharding:** Implement techniques like clustering or sharding in the database to distribute data across multiple servers for improved performance and scalability.
- **Monitor system performance for scalability adjustments:** Continuously monitor system performance metrics to identify bottlenecks and make necessary adjustments to maintain scalability as the application grows.



# CHAPTER 6

## DESIGN IMPLEMENTATION

### 6.1 FRONTEND DESIGN

#### 6.1.1 User Interface Design

- Create a simple and intuitive interface.
- **Main pages:** Home, Login, Registration, Dashboards, Quiz, Results.

##### 6.1.1.1 Wireframes for UI mockups:

- Sketch basic layouts for key pages.
- Ensure easy navigation and clear presentation of information.

##### 6.1.1.2 User Interaction Flow

###### 1. Registration/Login:

- Register with username, email, and password.
- Login with email and password.
- Redirect to the appropriate dashboard after login.

###### 2. Quiz Participation:

- Select a quiz from the available list.
- Answer questions with a timer.
- Submit answers and view results.

###### 3. Dashboard:

- **Quiz Creator:** Create, edit, and delete quizzes.
- **Participant:** View and take quizzes, see results.

#### 6.1.2 Component Architecture (React components)

1. **App Component:** Main wrapper.
2. **Header Component:** Navigation bar.
3. **Footer Component:** Footer section.
4. **Home Component:** Welcome page.
5. **Auth Components:** Login and registration forms.
6. **Dashboard Components:**
  - **QuizCreatorDashboard:** Quiz management tools.
  - **ParticipantDashboard:** Quiz list and progress.
7. **Quiz Components:**
  - **QuizList:** Display available quizzes.
  - **QuizQuestion:** Show individual questions.
  - **QuizResult:** Show results after submission.
8. **Common Components:** Buttons, forms, alerts.

### 6.1.3 State Management (Redux, Context API, etc.)

- Use **Redux** or **Context API** to manage global state.
- **Auth State:** Manage user authentication.
- **Quiz State:** Manage quizzes and questions.
- **Result State:** Manage quiz results.

### 6.1.4 API Integration (Backend endpoints)

#### 1. Authentication API:

- POST /register: Register a new user.
- POST /login: Authenticate a user.

#### 2. Quiz Management API:

- GET /quizzes: Get list of quizzes.
- POST /quizzes: Create a new quiz.
- PUT /quizzes/: Edit a quiz.
- DELETE /quizzes/: Delete a quiz.

#### 3. Quiz Participation API:

- GET /quizzes/: Get quiz details.
- POST /quizzes/ /submit: Submit quiz answers.

#### 4. Results API:

- GET /results: Get user results.
- GET /results/: Get specific quiz results.

## 6.2 BACKEND DESIGN

### 6.2.1 API Design

- **Endpoints:**
  1. POST /register: Register a new user.
  2. POST /login: Log in a user.
  3. GET /quizzes: Get all quizzes.
  4. POST /quizzes: Create a new quiz.
  5. PUT /quizzes/: Edit a quiz by its ID.
  6. DELETE /quizzes/: Delete a quiz by its ID.
  7. GET /quizzes/: Get details of a specific quiz.
  8. POST /quizzes/ /submit: Submit quiz answers.
  9. GET /results: Get results for the logged-in user.

10. GET /results/: Get results of a specific quiz.

## 6.2.2 Authentication and Authorization

### 1. User Registration and Login:

- Users register with username, email, and password.
- Users log in with email and password.

### 2. JWT Tokens

Use JWT tokens for session management, including them in the header of protected routes with a structure comprising metadata, user information (claims), and a signature to maintain integrity.

### 3. Role-Based Access Control (Optional)

- Quiz Creator: Can create, edit, and delete quizzes.
- Participant: Can take quizzes and view results.

## 6.2.3 Database Design and Integration

### 6.2.3.1 MongoDB Collections:

- **Users: Fields** - username, email, password, role.
- **Quizzes: Fields** - title, questions, createdBy, createdAt
- **Questions: Fields** - quizId, questionText, options, correctOption
- **Results: Fields** - userId, quizId, score, answers, completedAt.

### 6.2.3.2 CRUD Operations:

**CRUD** operations are fundamental to database management systems:

- **Create:** Add new data, e.g., registering a user on a website.
- **Read:** Fetch existing data, like retrieving user profile information upon login.
- **Update:** Update data, such as changing a user's profile picture or updating their password.
- **Delete:** Remove data, e.g., delete a user account and associated information.

# CHAPTER 7

## TESTING

### 7.1 TESTING STRATEGY

- **Purpose:** Define the approach to testing your MERN stack quiz application.
- **Explanation:** Discuss the types of testing (unit, integration, acceptance, performance) suitable for the application. Outline the testing tools, environments (development, staging, production), and methodologies (e.g., Agile, Scrum) used for testing.

### 7.2 UNIT TESTING

- **Purpose:** Verify the functionality of individual units or components.
- **Explanation:** Use Jest and React Testing Library for frontend React components. For backend Node.js APIs, utilize tools like Mocha or Jest with Supertest to test routes, controllers, and services. Ensure comprehensive coverage of business logic and edge cases.

### 7.3 INTEGRATION TESTING

- **Purpose:** Validate the interaction between different modules and components.
- **Explanation:** Employ tools like Supertest for API endpoints to verify data flow, validation, and integration with MongoDB. Test frontend-to-backend communication and ensure seamless interaction between React components and Node.js APIs.

# **CHAPTER 8**

## **DEPLOYMENT**

### **8.1 DEPLOYMENT STRATEGY**

The deployment strategy for the MERN quiz application outlines the methodical approach to transferring the application from development to production environments.

### **8.2 SETTING UP PRODUCTION ENVIRONMENT**

Setting up the production environment involves configuring servers and installing necessary software to host the MERN quiz application securely and efficiently.

### **8.3 CLOUD DEPLOYMENT**

Cloud deployment refers to hosting the MERN quiz application on cloud platforms like AWS, Azure, or Google Cloud, utilizing their services for scalability and accessibility.

### **8.4 MONITORING AND MAINTENANCE**

Monitoring and maintenance are crucial for ensuring the MERN quiz application's performance and reliability in production. Monitoring involves tracking metrics like response times and resource usage, while maintenance includes tasks such as backups and updates to sustain optimal operation.

# CHAPTER 9

## SYSTEM REQUIREMENTS

### 9.1 HARDWARE REQUIREMENTS

- **Device Name:** DESKTOP-NLI258U
- **Processor:** 12th Gen Intel(R) Core(TM) i3-1215U 1.20 GHz
- **RAM:** 8.00 GB
- **Keyboard:** 102 Keys enhanced

### 9.2 SOFTWARE REQUIREMENTS

- **Operating System:** Windows 11
- **Language:** JavaScript, CSS (Styling)
- **Client Side:**
- **Frontend Framework:** React.js.
- **Server Side:**
- **Backend Framework** - Node.js with Express.js
- **Database:** MongoDB
- **NPM Packages:**
- **Client Side:**
  1. **fontawesome:** npm install @fontawesome/fontawesome-free@^6.5.2
  2. **axios:** npm install axios
  3. **cors:** npm install cors
  4. **react:** npm install react
  5. **react-dom:** npm install react-dom
  6. **react-router-dom:** npm install react-router-dom
- **Server Side:**
  1. **bcrypt:** npm install bcrypt
  2. **cors:** npm install cors
  3. **express:** npm install express
  4. **jsonwebtoken:** npm install jsonwebtoken
  5. **mongoose:** npm install mongoose
  6. **nodemon:** npm install nodemon

### 9.3 LANGUAGE DESCRIPTION:

- **HTML:** Standard language for creating web pages using tags to define structure and content like headings, paragraphs, links, images, and forms.
- **CSS:** Language for styling HTML and XML documents, controlling layout, appearance, and design elements such as colors, fonts, and spacing.
- **Bootstrap:** Framework aiding responsive and visually appealing websites and apps with pre-designed CSS and JavaScript components.
- **JQuery:** Fast, small JavaScript library simplifying HTML document traversing, event handling, animation, and Ajax interactions for rapid web development.

- **Node.js:** Runtime environment enabling JavaScript code execution outside browsers, favored for server-side applications due to efficiency and scalability.
- **Express.js:** Minimal, flexible web application framework for Node.js, handling HTTP requests, routes, middleware, and database integration like MongoDB.
- **React.js:** JavaScript library by Facebook for building reusable UI components and managing state efficiently, using a declarative approach for dynamic web apps.
- **MongoDB:** NoSQL database system storing data in JSON-like BSON format, scalable, and suitable for managing diverse, unstructured data in modern web applications.

# **CHAPTER 10**

## **CONCLUSION**

### **10.1 PROJECT SUMMARY**

Provide a concise summary of the MERN Quiz App project, including its objectives, key features, and overall outcomes. Highlight the significance of the project and its relevance to the target audience.

### **10.2 FUTURE ENHANCEMENTS**

Discuss potential future enhancements and updates for the MERN Quiz App. Consider improvements in functionality, user experience, scalability, and performance. Outline new features or modules that could be added to further enhance the application.



# CHAPTER 11

## SAMPLE CODING

### FRONTEND

#### App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import '@fortawesome/fontawesome-free/css/all.min.css';
import Home from '../components/Home';
import Student from '../components/Student';
import Teacher from '../components/Teacher';
import TeacherSignup from '../components/TeacherSignup';
import TeacherLogin from '../components/TeacherLogin';
import StudentSignup from '../components/StudentSignup';
import StudentLogin from '../components/StudentLogin';
import StudentDashboard from './StudentDashboard';
import DashboardStu from './DashboardStu';
import ExamStu from './ExamStu';
import MarkStu from './MarkStu';
import TeacherDashboard from './TeacherDashboard';
import DashboardTea from './DashboardTea';
import ExamTea from './ExamTea';
import ExamRules from './ExamRules';
import QuestionList from './QuestionList';
import MarkList from './MarkList';
import ViewExamMarks from './ViewExamMarks';
import { StudentProvider } from '../Context/StudentContext';
import { TeacherProvider } from '../Context/TeacherContext';
import ViewExamTea from './ViewExamTea';
```

```

import LogoutSuccessPage from './LogoutSuccessPage';
import AddExamTea from './AddExamTea';
import QuestionTea from './QuestionTea';
import AddQuestionTea from './AddQuestionTea';
import SeleteCourseViewQuestionTea from './SelectCourseViewQuestionTea';
import ViewQuestionTea from './ViewQuestionTea';
import UpdateQuestionTea from './UpdateQuestionTea';
import UpdateExamTea from './UpdateExamTea';
function App() {
  return (
    <StudentProvider>
    <TeacherProvider>
    <Router>
    <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/studentsignup" element={<StudentSignup />} />
    <Route path="/studentlogin" element={<StudentLogin />} />
    <Route path="/teacher" element={<Teacher />} />
    <Route path="/student" element={<Student />} />
    <Route path="/teachersignup" element={<TeacherSignup />} />
    <Route path="/teacherlogin" element={<TeacherLogin />} />
    <Route path="/studentdashboard" element={<StudentDashboard />} />
    <Route path="/dashboardstu" element={<DashboardStu />} />
    <Route path="/examstu" element={<ExamStu />} />
    <Route path="/markstu" element={<MarkStu />} />
    <Route path="/teacherdashboard" element={<TeacherDashboard />} />
    <Route path="/dashboardtea" element={<DashboardTea />} />
    <Route path="/examtea" element={<ExamTea />} />
    <Route path="/examrules/:examId/:examName" element={<ExamRules />} /> <Route
    path="/questionlist/:examId/:examName" element={<QuestionList />}/>
  )
}

```

```

<Route path='/marklist/:examId/:examName' element={<MarkList />} /> <Route
path='/viewexammarks' element={<ViewExamMarks />} />
<Route path='/viewexamtea' element={<ViewExamTea />} />
<Route path='/logoutsuccesspage' element={<LogoutSuccessPage />} />
<Route path='/addexamtea' element={<AddExamTea />} />
<Route path="/updateexamtea/:id" element={<UpdateExamTea />} />
<Route path='/questiontea' element={<QuestionTea />} />
<Route path='/addquestiontea' element={<AddQuestionTea />} />
<Route path="/seletecourseviewquestiontea" element={<SeleteCourseViewQuestionTea />} />
<Route path="/viewquestiontea" element={<ViewQuestionTea />} />
<Route path="/updatequestiontea/:id" element={<UpdateQuestionTea />} /> </Routes>
</Router>
</TeacherProvider>
</StudentProvider>
);}

```

export default App;

### **TeacherSignup.jsx**

```

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import '../styles/TeacherSignup.css';
function TeacherSignup() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errors, setErrors] = useState({});
  const [signupError, setSignupError] = useState("");
  const navigate = useNavigate();
  const validateForm = () => {
    const newErrors = {};
    if (!name.trim()) newErrors.name = 'Name is required';

```

```

    if (!email.trim()) {
      newErrors.email = 'Email is required';
    } else if (!/^\S+@\S+\.\S+/.test(email)) {
      newErrors.email = 'Email is invalid'; }
    if (!password.trim()) newErrors.password = 'Password is required';
    return newErrors; };

const handleSubmit = async (e) => {
  e.preventDefault();
  const formErrors = validateForm();
  if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
    return; }
  setErrors({ });
  setSignupError("");
  try {
    const response = await fetch('http://localhost:5000/teacher/register',
      method: 'POST',
      headers: {
        'Content-Type': 'application/json', },
      body: JSON.stringify({ name, email, password }), });
    if (response.ok) {
      navigate('/teacherlogin'); }
    else if (response.status === 409) {
      const data = await response.json();
      setSignupError(data.message);
    } else {
      setSignupError('Signup failed'); }}
  catch (error) {
    setSignupError('An error occurred. Please try again.');
```

```

const handleLoginClick = () => {

```

```

        navigate('/teacherlogin'); });

return (
  <div className="teachersignup-container">
    <h2>Teacher Signup</h2>
    <form className="teachersignup-form" onSubmit={handleSubmit}>
      <input type="text"
        name="name" placeholder="Name" value={name}
        onChange={(e) => setName(e.target.value)} autoComplete="name" />
      {errors.name && <div className="error">{errors.name}</div>}
      <input type="email" name="email" placeholder="Email" value={email} onChange={(e) =>
        setEmail(e.target.value)} autoComplete="email"/>
      {errors.email && <div className="error">{errors.email}</div>}
      <input type="password" name="password" placeholder="Password" value={password}
        onChange={(e) => setPassword(e.target.value)}
        autoComplete="new-password" />
      {errors.password && <div className="error">{errors.password}</div>}
      <button type="submit">Sign up</button>
      {signupError && <div className="signup-error">{signupError}</div>}
      {signupError === 'You already have an account.' && (
        <button className="login-button" onClick={handleLoginClick}>Login
        </button> )}</form> </div> ); }

export default TeacherSignup;

```

### **TeachereLogin.jsx**

```

import React, { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { TeacherContext } from '../Context/TeacherContext';
import '../styles/TeacherLogin.css';

function TeacherLogin() {
  const { setTeacher } = useContext(TeacherContext);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errors, setErrors] = useState({});

```

```

const [loginError, setLoginError] = useState("");
const navigate = useNavigate();
const validateForm = () => {
const newErrors = { };
    if (!email.trim()) {
        newErrors.email = 'Email is required';
    } else if (!/^S+@\S+\.\S+/.test(email)) {
        newErrors.email = 'Email is invalid'; }
    if (!password.trim()) newErrors.password = 'Password is required';
    return newErrors; };
const handleSubmit = async (e) => { // (e) is the event
e.preventDefault();
const formErrors = validateForm();
if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
return; }
    setErrors({ });
    setLoginError("");
try {
const response = await fetch('http://localhost:5000/teacher/login',
    method: 'POST',
    headers: {
        'Content-Type': 'application/json', },
    body: JSON.stringify({ email, password }), });
if (response.ok) {
    const data = await response.json();
    localStorage.setItem('token', data.token);
    setTeacher(data); // TeacherContext
    navigate('/teacherDashboard'); }
    else if (response.status === 404) {

```

```

        const data = await response.json();
        setLoginError(data.message); }
    else {
        setLoginError('Login failed'); } }
    catch (error) {
        setLoginError('An error occurred. Please try again. ');
    } } };
const handleSignupClick = () => { navigate('/teachersignup'); };
return (
    <div className="teacherlogin-container">
    <h2>Teacher Login</h2>
    <form className="teacherlogin-form" onSubmit={handleSubmit}>
    <input type="email" name="email" placeholder="Email" value={email} onChange={(e) =>
    setEmail(e.target.value)} autoComplete="email" /> {errors.email} && <div
    className="error">{errors.email}</div>
    <input type="password" name="password" placeholder="Password" value={password}
    onChange={(e) => setPassword(e.target.value)}
    autoComplete="new-password" />
    {errors.password} && <div className="error">{errors.password}</div>
    <button type="submit">Log in</button>
    {loginError} && <div className="login-error">{loginError}</div>
    {loginError === "Don't have an account?" && (
    <button className="signup-button" onClick={handleSignupClick}> Sign up
    </button> )} </form> </div> ); }
export default TeacherLogin;

```

### **StudentSignup.jsx**

```

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import '../styles/StudentSignup.css';
function StudentSignup() {
    const [name, setName] = useState("");

```

```

const [email, setEmail] = useState("");
const [password, setPassword] = useState("");
const [errors, setErrors] = useState({ });
const [signupError, setSignupError] = useState("");
const navigate = useNavigate();
const validateForm = () => { const newErrors = { };
    if (!name.trim()) newErrors.name = 'Name is required';
    if (!email.trim()) {
        newErrors.email = 'Email is required';
    } else if (!/^S+@\S+\.\S+/.test(email)) {
        newErrors.email = 'Email is invalid' }
    if (!password.trim()) newErrors.password = 'Password is required';
    return newErrors; };
const handleSubmit = async (e) => { e.preventDefault();
const formErrors = validateForm();
if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
    return; }
setErrors({ });
setSignupError("");
try {
    const response = await fetch('http://localhost:5000/register',
        method: 'POST', headers: { 'Content-Type': 'application/json', },
        body: JSON.stringify({ name, email, password }), });
    if (response.ok) { navigate('/studentlogin'); }
    else if (response.status === 409) { const data = await response.json();
        setSignupError(data.message); }
    else {
        setSignupError('Signup failed'); } }
catch (error) {

```



```

        setSignupError('An error occurred. Please try again.');
```

```

const handleLoginClick = () => { navigate('/studentlogin'); };

return (
  <div className="studentsignup-container"> <h2>Student Signup</h2>
  <form className="studentsignup-form" onSubmit={handleSubmit}>
    <input type="text" name="name" placeholder="Name" value={name}
    onChange={(e) => setName(e.target.value)} autoComplete="name" />
    {errors.name && <div className="error">{errors.name}</div>}
    <input type="email" name="email" placeholder="Email" value={email}
    onChange={(e) => setEmail(e.target.value)} autoComplete="email" />
    {errors.email && <div className="error">{errors.email}</div>}
    <input type="password" name="password" placeholder="Password"
    value={password} onChange={(e) => setPassword(e.target.value)}
    autoComplete="new-password" />
    {errors.password && <div className="error">{errors.password}</div>}
    <button type="submit">Sign up</button>
    {signupError && <div className="signup-error">{signupError}</div>}
    {signupError === 'You already have an account.' && (
      <button className="login-button" onClick={handleLoginClick}>Log in
      </button> )} </form> </div> ); } export default StudentSignup;

```

### **StudentLogin.jsx**

```

import React, { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import { StudentContext } from '../Context/StudentContext';
import '../styles/StudentLogin.css';

function StudentLogin() {
  const { setStudent } = useContext(StudentContext);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errors, setErrors] = useState({ });

```

```

const [loginError, setLoginError] = useState("");

const navigate = useNavigate(); const validateForm = () => { const newError {}; if
(!email.trim()) {
    newErrors.email = 'Email is required';
} else if (!/\S+@\S+\.\S+/.test(email)) {
    newErrors.email = 'Email is invalid'; }
    if (!password.trim()) newErrors.password = 'Password is required';
return newErrors };

const handleSubmit = async (e) => { e.preventDefault();
const formErrors = validateForm();
    if (Object.keys(formErrors).length > 0) {
        setErrors(formErrors);
        return; } setErrors({}); setLoginError("");
try {
const response = await fetch('http://localhost:5000/student/login', {
method: 'POST', headers:
'Content-Type': 'application/json', }, body: JSON.stringify({ email, password }), }); if
(response.ok) {
    const data = await response.json();
    localStorage.setItem('token', data.token);
    setStudent(data); navigate('/studentdashboard'); } else {
    const data = await response.json();
    setLoginError(data.message || 'Login failed'); }
} catch (error) {
    console.error('Login error:', error);
    setLoginError('An error occurred. Please try again.');
```

```

<input type="email" name="email" placeholder="Email" value={email}
onChange={(e) => setEmail(e.target.value)} autoComplete="email"
aria-label="Email" />
{errors.email && <div className="error">{errors.email}</div>}
<input type="password" name="password" placeholder="Password"
value={password} onChange={(e) => setPassword(e.target.value)}
autoComplete="new-password" aria-label="Password" />
{errors.password && <div className="error">{errors.password}</div>}
<button type="submit">Log in</button>
{loginError && <div className="login-error">{loginError}</div>}
{loginError === "Don't have an account?" &
<button type="button" className="signup-button" onClick={handleSignupClick}> Sign up
</button> )} </form> </div> ); }
export default StudentLogin;

```

### **AddExamTea.jsx**

```

import React from 'react';
import { useNavigate } from 'react-router-dom';
import '../styles/DashboardTea.css';
import TeacherNavSidebarOpen from './TeacherNavSidebarOpen';
import QuizAppFooter from './QuizAppFooter';
export default function AddExamTea() {
  const navigate = useNavigate();
  const handleSubmit = async (event) => {
    event.preventDefault();
    const formData = new FormData(event.target);
    try {
      const response = await fetch('http://localhost:5000/api/addexamtea', {
        method: 'POST', headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          examName: formData.get('examName'),

```

```

        totalQuestions: parseInt(formData.get('totalQuestions')),
        totalMarks: parseInt(formData.get('totalMarks')), )),  });
if (response.ok) {
    const data = await response.json();
    console.log('Exam added:', data);
    navigate('/viewexamtea'); }
else {
    console.error('Failed to add exam'); } }
catch (error) { console.error('Error adding exam:', error ) };
return (
<div className='dashboardtea-container'>
<TeacherNavSidebarOpen /> <div className="addexamtea">
<h1>ADD EXAM</h1> <form onSubmit={handleSubmit}>
<label>Exam Name
<input type="text" name="examName" placeholder="Express.js" required />
</label> <label>Total Questions
<input type="number" name="totalQuestions" placeholder="5" required />
</label> <label>Total Marks
<input type="number" name="totalMarks" placeholder="5" required />
</label> <button type="submit">ADD</button> </form> </div>
<QuizAppFooter /> </div> );

```

### **ViewExamTea.jsx**

```

import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import '../styles/ExamStu.css'
import TeacherNavSidebarOpen from './TeacherNavSidebarOpen';
import QuizAppFooter from './QuizAppFooter';
export default function ViewExamTea() {
    const [exams, setExams] = useState([]);
    useEffect(() => {

```

```

    fetchExams(); }, []);
const fetchExams = async () => {
  try {
    const response = await fetch('http://localhost:5000/api/viewexamtea');
    const data = await response.json();
    setExams(data);
  } catch (error) {
    console.error('Error fetching exams:', error); } };
const handleDelete = async (examId) => {
  try {
    const response = await fetch(`http://localhost:5000/api/viewexamtea/${examId}`, { method:
    'DELETE', });
    if (response.ok) { setExams(exams.filter(exam => exam._id !== examId)); } else {
    console.error('Failed to delete exam'); } } catch (error) {
    console.error('Error deleting exam:', error); } };
  return (
    <div className="examstu-container">
      <TeacherNavSidebarOpen />
      <div className='examstu-body'>
        <div className='examstu-courses'>
          <span>Exams</span></div><table className='examstu-table'><thead>
          <tr><th>Exam Name</th> <th>Total Questions</th> <th>Total Marks</th>
          <th>Edit</th> <th>Delete</th></tr> </thead>
          <tbody>
            {exams.map(exam => (
              <tr key={exam._id}> <td>{exam.examName}</td>
              <td>{exam.totalQuestions}</td> <td>{exam.totalMarks}</td> <td>
                <Link to={` /updateexamtea/${exam._id}`}>
                  <button className='delete-exam-btn'>
                    <i className="fa" id='edit-exam-icon'>&#xf044;</i> </button> </Link>

```

```

</td> <td>
<buttonclassName='delete-exam-btn'onClick={() => handleDelete(exam._id)}>
<i className="fa" id='delete-exam-icon'>&#xf014;</i> </button> </td> </tr>
))} </tbody> </table> </div> <QuizAppFooter /> </div> ); }

```

### UpdateExamTea.jsx

```

import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import TeacherNavSidebarOpen from './TeacherNavSidebarOpen';
import QuizAppFooter from './QuizAppFooter';
export default function UpdateExamTea() {
  const { id } = useParams();
  const navigate = useNavigate();
  const [exam, setExam] = useState({
    examName: "", totalQuestions: "", totalMarks: "" });
  useEffect(() => {
    const fetchExam = async () => {
      try {
        const response = await fetch(`http://localhost:5000/api/exams/${id}`);
        const data = await response.json();
        setExam(data);
      } catch (error) {
        console.error('Error fetching exam:', error); } };
    fetchExam(); }, [id]);
  const handleChange = (e) => {
    const { name, value } = e.target;
    setExam(prevExam => ({ ...prevExam, [name]: value })); };
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await fetch(`http://localhost:5000/api/updateexamtea/${id}`, {

```

```

    method: 'PUT',
    headers: {
      'Content-Type': 'application/json' },
    body: JSON.stringify(exam) });
if (response.ok) { navigate('/viewexamtea'); } else {
  console.error('Failed to update exam'); } } catch (error) {
  console.error('Error updating exam:', error); } };
return (
  <div className='dashboardtea-container'>
    <TeacherNavSidebarOpen />
    <div className="addexamtea">
      <h1>Update Exam</h1>
      <form onSubmit={handleSubmit}>
        <label>Exam name
        <input type="text" name="examName" value={exam.examName} onChange={handleChange}
        required /> </label> <label>Total Question
        <input type="number" name="totalQuestions" value={exam.totalQuestions}
        onChange={handleChange} required /> </label> <label>Total Marks
        <input type="number" name="totalMarks" value={exam.totalMarks}
        onChange={handleChange} required /> </label>
        <button type="submit">Update</button> </form> </div> <QuizAppFooter /> </div> ); }

```

### **ViewQuestionTea.jsx**

```

import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import TeacherNavSidebarOpen from './TeacherNavSidebarOpen';
import QuizAppFooter from './QuizAppFooter';
export default function UpdateQuestionTea() {
  const { id } = useParams(); // Get question ID from URL
  const navigate = useNavigate();
  const [questionData, setQuestionData] = useState({ examId: "", questionText: "",
    marks: "", options: ["", "", ""], answer: "" });

```

```

const [exams, setExams] = useState([]);
useEffect(() => {
  const fetchQuestion = async () => {
    try {
      const response = await fetch(`http://localhost:5000/api/questions/${id}`);
      if (!response.ok) {
        throw new Error('Failed to fetch question'); }
      const question = await response.json();
      setQuestionData({
        examId: question.examId,
        questionText: question.questionText,
        marks: question.marks,
        options: question.options,
        answer: question.answer, });
    } catch (error) {
      console.error('Error fetching question:', error.message); } };
    const fetchExams = async () => {
      try {
        const response = await fetch('http://localhost:5000/api/viewexamtea');
        if (!response.ok) {
          throw new Error('Failed to fetch exams'); }
        const examsData = await response.json();
        setExams(examsData);
      } catch (error) {
        console.error('Error fetching exams:', error); } };
      fetchQuestion();
      fetchExams();
    }, [id]);
    const handleChange = (e) => {
      const { name, value } = e.target;

```



```

setQuestionData((prevData) => ({
  ...prevData,
  [name]: value, })); });
const handleOptionChange = (index, value) => {
  const newOptions = [...questionData.options];
  newOptions[index] = value;
  setQuestionData((prevData) => ({ ...prevData, options: newOptions, })); });
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await fetch(`http://localhost:5000/api/updatequestiontea/${id}`, { method:
    'PUT', headers: { 'Content-Type': 'application/json', },
    body: JSON.stringify(questionData), });
    if (response.ok) {
      navigate(`/viewquestiontea?examId=${questionData.examId}`);
    } else { const errorData = await response.json();
      console.error('Failed to update question:', errorData); }
    } catch (error) {
      console.error('Error updating question:', error); } } };
return (
  <div className='dashboardtea-container'>
    <TeacherNavSidebarOpen /> <div className="addexamtea">
      <h1>Update Question</h1> <form onSubmit={handleSubmit}> <label>Exam
      <select name="examId" value={questionData.examId}
      onChange={handleChange} required >
        <option value="">Select Exam</option>
        {exams.map((exam) => (
          <option key={exam._id} value={exam._id}>{exam.examName}</option> ))}
      </select>
    </label>
  </div>
  </div>

```

```

<label>Question
<textarea name="questionText" value={questionData.questionText}
onChange={handleChange} placeholder="Enter question text" required>
</textarea>
</label> <label> Marks <input type="number" name="marks"
value={questionData.marks} onChange={handleChange}
placeholder="Enter marks" required />
</label> <label> Option 1 <input type="text" value={questionData.options[0]}
onChange={(e) => handleOptionChange(0, e.target.value)}
placeholder="Option 1" required />
</label> <label> Option 2 <input type="text" value={questionData.options[1]}
onChange={(e) => handleOptionChange(1, e.target.value)}
placeholder="Option 2" required />
</label> <label> Option 3 <input type="text" value={questionData.options[2]}
onChange={(e) => handleOptionChange(2, e.target.value)}
placeholder="Option 3" required /> </label>
<label>Answer <select name="answer" value={questionData.answer}
onChange={handleChange} required >
<option value="">Select Answer</option>
<option value="option1">Option 1</option>
<option value="option2">Option 2</option>
<option value="option3">Option 3</option> </select>
</label> <button type="submit">Update</button> </form> </div>
<QuizAppFooter /> </div> ); }

```

### **QuestionList.jsx**

```

import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import '../styles/Question.css';
import QuizAppFooter from './QuizAppFooter';
import StudentNavSidebarOpen from './StudentNavSisebarOpen';

```

```

import Timer from './Timer';

function QuestionList() {
  const { examId, examName } = useParams();
  const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);
  const [questions, setQuestions] = useState([]);
  const [userAnswers, setUserAnswers] = useState([]);
  const [startTime, setStartTime] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    setStartTime(new Date());
    setQuestions([]);
    setUserAnswers([]);
    const fetchQuestions = async () => {
      try {
        const response = await
        fetch(`http://localhost:5000/api/viewquestions?examId=${examId}`);
        const data = await response.json();
        setQuestions(data);
        setUserAnswers(new Array(data.length).fill(undefined)); // Initialize userAnswers
        array
      } catch (error) {
        console.error('Error fetching questions:', error);
      }
      fetchQuestions();
    }, [examId]);

    const onPrev = () => {
      if (currentQuestionIndex > 0) {
        setCurrentQuestionIndex(currentQuestionIndex - 1);
      }
    };

    const onNext = () => {
      if (currentQuestionIndex < questions.length - 1) {
        setCurrentQuestionIndex(currentQuestionIndex + 1);
      }
    };
  });
}

```

```

const onSubmit = () => {
  const endTime = new Date();
  const duration = Math.floor((endTime - startTime) / 1000);
  let totalMarks = 0;
  userAnswers.forEach((answer, index) => {
    const question = questions[index];
    const correctAnswer = question?.answer
const correctAnswerIndex=question.options[parseInt(correctAnswer.replace
('option', '')) - 1];
    console.log(`Question ${index + 1}: Correct Answer = ${correctAnswerIndex}, User Answer =
    ${answer}`);
    if (answer === correctAnswerIndex) {
      totalMarks += question?.marks || 0; } });
    console.log('Total Marks:', totalMarks);
    console.log('User Answers:', userAnswers);
    console.log('Questions:', questions);
    navigate(`/marklist/${examId}/${examName}`, {
      state: { totalMarks, startTime: startTime.toISOString(), duration,
        examName } }); });
const onSelect = (e) => {
  const newAnswers = [...userAnswers];
  newAnswers[currentQuestionIndex] = e.target.value;
  setUserAnswers(newAnswers); });
const onTimeUp = () => {
  alert('Time is up!');
  onSubmit(); });
if (!questions || questions.length === 0) {
  return <div>No questions available</div>; }
const question = questions[currentQuestionIndex];
const checked = userAnswers[currentQuestionIndex];

```

```

return (
  <div className="question-container"> <StudentNavSidebarOpen />
  <div className='question-container-body'>
    <div className='question-heading'> <h1>{examName} Quiz</h1>
    {question}&&<h2>Question{currentQuestionIndex+1} of
    {questions.length}:</h2>}
    <Timer initialMinutes={5} initialSeconds={0} onTimeUp={onTimeUp} />
  </div>
  {question ? (
    <div className='question'>
      <h3>{question.questionText}</h3>
      <ul key={question._id}>
        {question.options.map((option, i) => (
          <li key={i}>
            <input type="radio" value={option}
            name={`question${currentQuestionIndex}`} id={`q${i}-option`}
            checked={checked === option} onChange={onSelect} />
            <label htmlFor={`q${i}-option`} >{option}</label>
            <div className={`check ${checked === option ? 'checked' : ''}`></div></li>
          ))) </ul> </div> ) : (
        <div>No question available</div> )}
      <div className='prev-next-btn'>
        {currentQuestionIndex > 0 && (
          <button className='btn-prev' onClick={onPrev}>
            <i id='prev-icon'>&laquo;</i> Prev </button> )}
        {currentQuestionIndex < questions.length - 1 && (
          <button className='btn-next' onClick={onNext}>
            Next <i id='next-icon'>&raquo;</i> </button> )}
        {currentQuestionIndex === questions.length - 1 && (
          <button className='btn-submit' onClick={onSubmit}> Submit </button> )}

```

```

</div> </div> <QuizAppFooter /> </div> ); }
export default QuestionList;

MarkList.jsx
import React from 'react';
import { useLocation } from 'react-router-dom';
import '../styles/Marks.css';
import QuizAppFooter from './QuizAppFooter';
import StudentNavSidebarOpen from './StudentNavSisebarOpen';

function MarkList() {
  const location = useLocation();
  const { examName, totalMarks, startTime, duration } = location.state || {
    examName: 'Unknown',
    totalMarks: 0, startTime: new Date().toISOString(), duration: 0 };
  const startDate = new Date(startTime);
  const durationMinutes = Math.floor(duration / 60);
  const durationSeconds = duration % 60;
  return (
    <div className="marks-container">
      <StudentNavSidebarOpen /> <div className='marks-body'>
        <div className='marks-courses'> <span>View Marks</span> </div>
        <table className='marks-table'> <thead>
          <tr> <th>Exam Name</th> <th>Total Marks</th> <th>Exam Date</th>
            <th>Duration</th> </tr> </thead> <tbody>
            <tr> <td>{examName}</td> <td>{totalMarks}</td>
              <td>{startDate.toLocaleString()}</td>
              <td>`${durationMinutes}m ${durationSeconds}s`</td>
            </tr> </tbody> </table> </div> <QuizAppFooter /> </div> ); }
export default MarkList;

```

## BACKEND

### index.js

```

const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const Student = require('./models/Student');
const Teacher = require('./models/Teacher');
const Exam = require('./models/Exam'); // Mongoose model
const Question = require('./models/Question');
const app = express();
const port = process.env.PORT || 5000;
app.use(cors());
app.use(express.json());
mongoose.connect('mongodb://localhost:27017/students', { useNewUrlParser: true,
useUnifiedTopology: true });
const verifyToken = (req, res, next) => {
  const token = req.headers['x-auth-token'];
  if (!token) return res.status(401).json({ message: 'No token, authorization denied' });
  try {
    const decoded = jwt.verify(token, 'secretKey');
    req.user = decoded;
    next(); } catch (e) {
    res.status(400).json({ message: 'Token is not valid' }); } };
app.get('/studentprofile', verifyToken, async (req, res) => {
  try {
    const student = await Student.findById(req.user.id).select('-password');
    res.json(student); } catch (error) {
    res.status(500).json({ message: 'Server Error' }); } });
app.get('/teacherprofile', verifyToken, async (req, res) => {
  try {

```

```

    const teacher = await Teacher.findById(req.user.id).select('-password');
    res.json(teacher); } catch (error) {
    res.status(500).json({ message: 'Server Error' }); } });
app.post('/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const existingStudent = await Student.findOne({ email });
    if (existingStudent)
    return res.status(409).json({ message: 'You already have an account.' });}
    const hashedPassword = await bcrypt.hash(password, 10);
    const newStudent = new Student(
    { name, email, password: hashedPassword}); await newStudent.save();
    res.status(201).send('Student registered'); }
  catch (error) {
    res.status(400).send(error.message); } });
app.post('/student/login', async (req, res) => {
  try {
    const { email, password } = req.body
    const student = await Student.findOne({ email });
    if (!student) {
      return res.status(404).json({ message: "Don't have an account?" }); }
    const isMatch = await bcrypt.compare(password, student.password);
    if (!isMatch) {
      return res.status(400).send('Invalid credentials'); }
    const token = jwt.sign({ id: student._id }, 'secretKey', { expiresIn: '1h' }); res.json({
token });
    } catch (error) { res.status(400).send(error.message); } });
app.get('/api/totalStudents', async (req, res) => {
  try {
    const totalStudents = await Student.countDocuments();

```



```

    res.json({ totalStudents }); } catch (err) {
    console.error('Error fetching total students:', err);
    res.status(500).json({ error: 'Failed to fetch total students' } } });
app.post('/teacher/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const existingTeacher = await Teacher.findOne({ email });
    if (existingTeacher) {
      return res.status(409).json({ message: 'You already have an account.' } }); } const
hashedPassword = await bcrypt.hash(password, 10);
    const newTeacher = new Teacher
    ({ name, email, password: hashedPassword });
    await newTeacher.save(); res.status(201).send("Teacher registered"); }
  catch (error) {
    res.status(400).send(error.message); } });
app.post('/teacher/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const teacher = await Teacher.findOne({ email });
    if (!teacher) {
      return res.status(404).json({ message: "Don't have an account?" }); } const isMatch
= await bcrypt.compare
    (password, teacher.password);
    if (!isMatch) {
      return res.status(400).send('Invalid credentials'); }
    const token = jwt.sign({ id: teacher._id }, 'secretKey', { expiresIn: '1h' });
    res.json({ token }); } catch (error) {
    res.status(400).send(error.message); } });
app.post('/api/addexamtea', async (req, res) => {
  try {

```

```

    const { examName, totalQuestions, totalMarks } = req.body;
    const newExam = new Exam({ examName, totalQuestions, totalMarks, });
    await newExam.save();
    res.status(201).json(newExam); } catch (err) {
    console.error('Error adding exam:', err);
    res.status(500).json({ error: 'Failed to add exam' }); } });

app.get('/api/viewexamtea', async (req, res) => {
  try {
    const exams = await Exam.find();
    res.json(exams); } catch (err) {
    console.error('Error fetching exams:', err);
    res.status(500).json({ error: 'Failed to fetch exams' }); } });

app.get('/api/exams/:id', async (req, res) => {
  try {
    const exam = await Exam.findById(req.params.id);
    res.json(exam); } catch (err) {
    console.error('Error fetching exam:', err);
    res.status(500).json({ error: 'Failed to fetch exam' }); } });

app.put('/api/updateexamtea/:id', async (req, res) => {
  try {
    const { examName, totalQuestions, totalMarks } = req.body;
    const updatedExam = await Exam.findByIdAndUpdate(req.params.id, {
      examName, totalQuestions, totalMarks, }, { new: true });
    if (updatedExam) {
      res.json(updatedExam); } else {
      res.status(404).json({ error: 'Exam not found' }); } } catch (err) {
      console.error('Error updating exam:', err);
      res.status(500).json({ error: 'Failed to update exam' }); } });

app.delete('/api/viewexamtea/:id', async (req, res) => {
  try {

```

```

const examId = req.params.id;
const result = await Exam.findByIdAndDelete(examId);
if (result) {
  res.status(200).json({ message: 'Exam deleted successfully' }); } else {
  res.status(404).json({ error: 'Exam not found' }); } } catch (err) {
console.error('Error deleting exam:', err);
res.status(500).json({ error: 'Failed to delete exam' }); } });
pp.get('/api/totalExams', async (req, res) => {
try {
  const totalExams = await Exam.countDocuments();
  res.json({ totalExams }); } catch (err) {
  console.error('Error fetching total exams:', err);
  res.status(500).json({ error: 'Failed to fetch total exams' }); } });
app.post('/api/addquestions', async (req, res) => {
  const { examId, questionText, marks, option1,
    option2, option3, answer } = req.body;
try {
  const newQuestion = new Question({ examId, questionText, marks,
    options: [option1, option2, option3], answer, });
  await newQuestion.save();
  res.status(201).json(newQuestion); } catch (err) {
  console.error('Error adding question:', err);
  res.status(500).json({ error: 'Failed to add question' }); } });
app.get('/api/viewquestions', async (req, res) => {
try {
  const { examId } = req.query; if (!examId) {
return res.status(400).json({ error: 'examId is required' }); }
  const questions = await Question.find({ examId });
  res.json(questions); } catch (err) {
  console.error('Error fetching questions:', err);

```

```

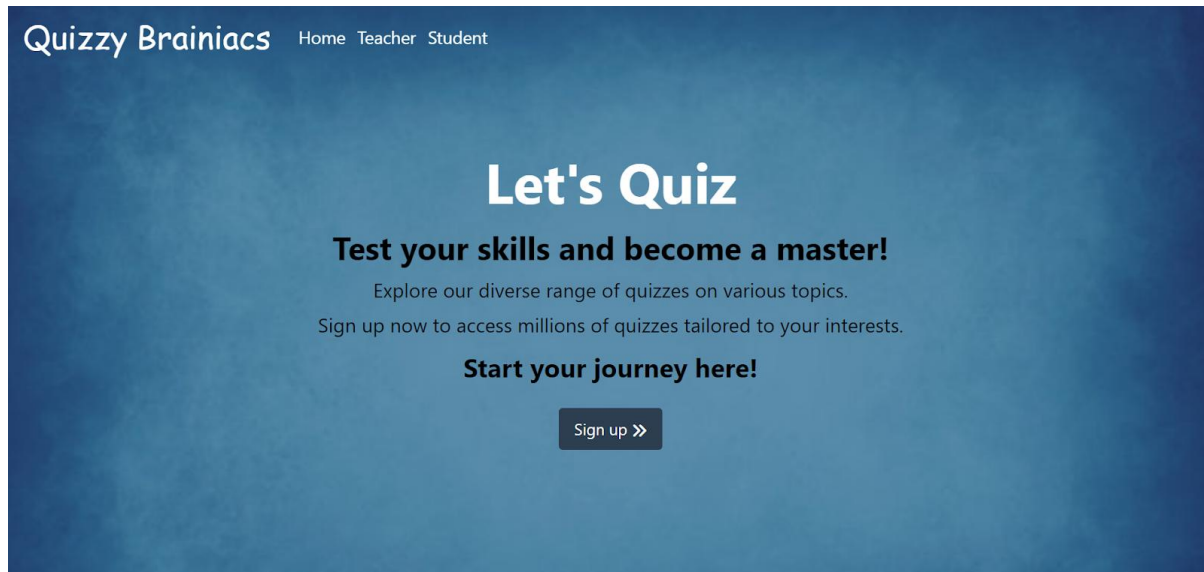
res.status(500).json({ error: 'Failed to fetch questions' }); } });
app.get('/api/questions/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const question = await Question.findById(id);
    if (!question) {
      return res.status(404).json({ error: 'Question not found' }); }
    res.json(question); } catch (error) {
      console.error('Error fetching question:', error);
      res.status(500).json({ error: 'Failed to fetch question' }); } });
app.put('/api/updatequestiontea/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { examId, questionText, marks, options, answer } = req.body;
    if (!examId || !questionText || !marks || !options || !answer) {
      return res.status(400).json({ error: 'All fields required' }); }
    const updatedQuestion = await Question.findByIdAndUpdate( id,
      { examId, questionText, marks, options, answer }, { new: true } );
    if (!updatedQuestion) {
      return res.status(404).json({ error: 'Question not found' }); }
    res.json(updatedQuestion); } catch (err) {
      console.error('Error updating question:', err);
      res.status(500).json({ error: 'Failed to update question' }); } });
app.delete('/api/viewquestions/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const question = await Question.findByIdAndDelete(id);
    if (!question) {
      return res.status(404).json({ error: 'Question not found' }); }
    res.status(204).send(); } catch (err) {

```

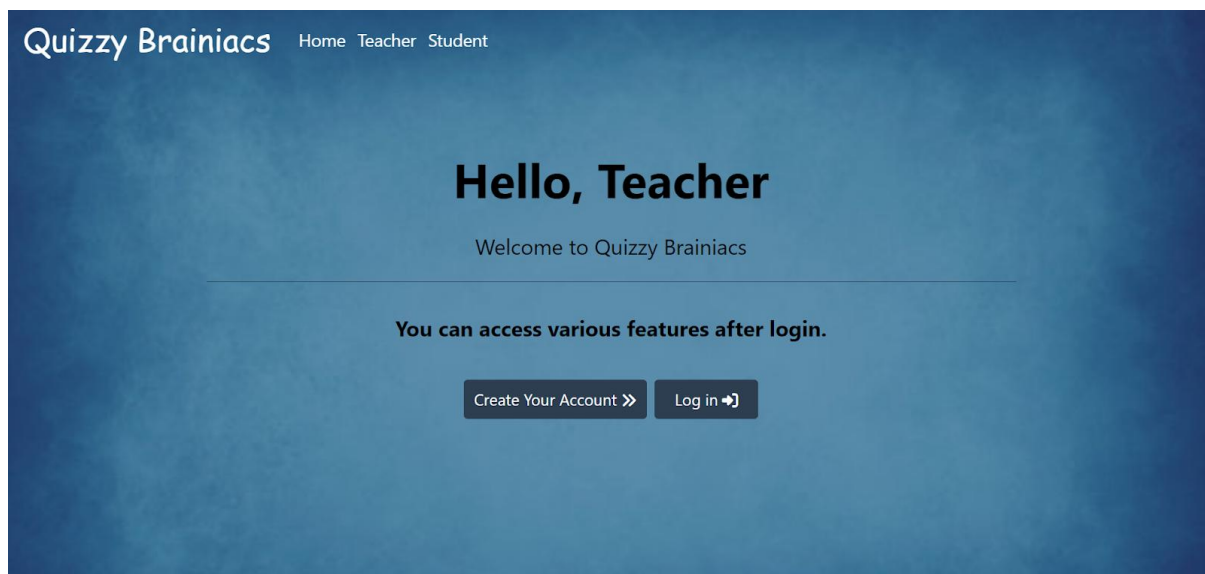
```
    console.error('Error deleting question:', err);
    res.status(500).json({ error: 'Failed to delete question' }); } });
app.get('/api/totalQuestions', async (req, res) => {
  try {
    const totalQuestions = await Question.countDocuments();
    res.json({ totalQuestions }); } catch (err) {
    console.error('Error fetching total questions:', err);
    res.status(500).json({ error: 'Failed to fetch total questions' }); } });
app.listen(port, () => {
  console.log(`Server running on port ${port}`); });
```

## CHAPTER 12

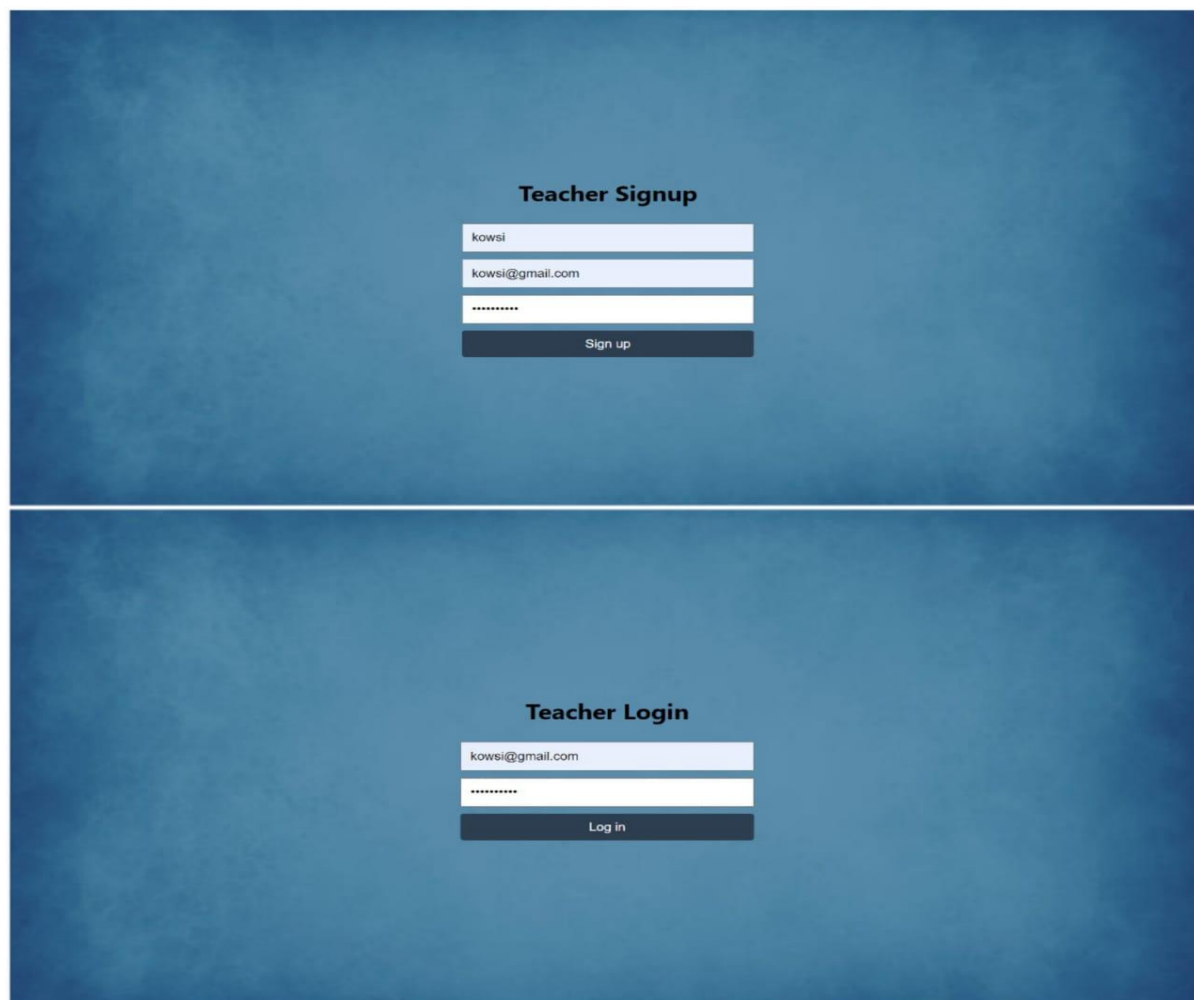
### SAMPLE SCREENSHOTS



**Fig: 12.1 QUIZZY BRAINIACS HOME PAGE**



**Fig: 12.2 TEACHER PAGE**



The image displays two separate form sections on a blue background. The top section is titled "Teacher Signup" and contains three input fields: a text field with "kowsi", an email field with "kowsi@gmail.com", and a password field with masked characters. Below these is a dark "Sign up" button. The bottom section is titled "Teacher Login" and contains two input fields: an email field with "kowsi@gmail.com" and a password field with masked characters. Below these is a dark "Log in" button.

**Teacher Signup**

kowsi

kowsi@gmail.com

\*\*\*\*\*

Sign up

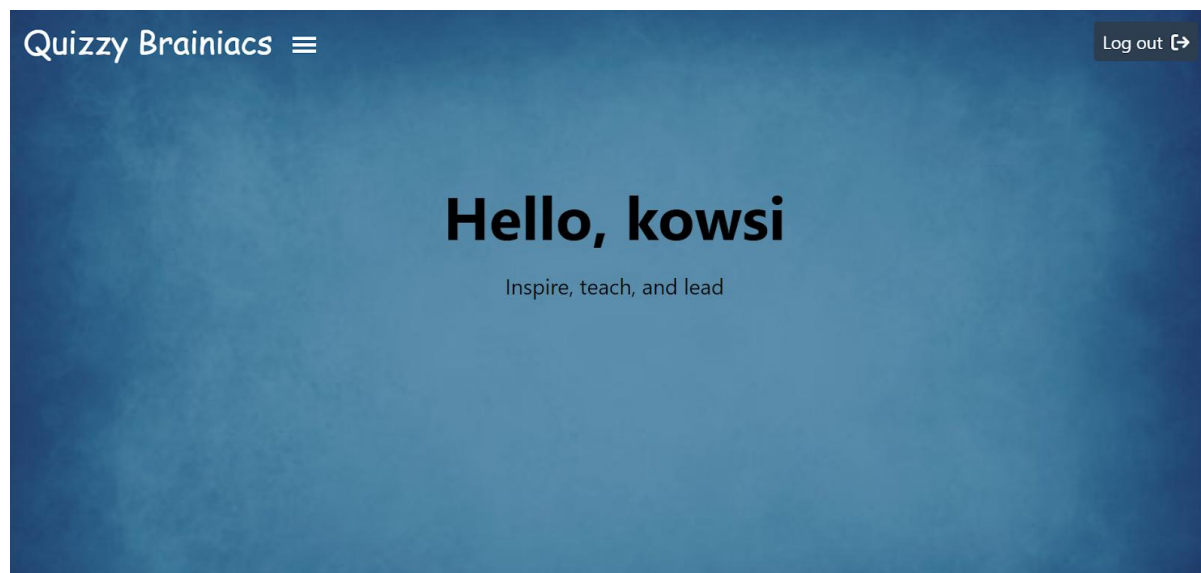
**Teacher Login**

kowsi@gmail.com

\*\*\*\*\*

Log in

**Fig: 12.3 TEACHER SIGN UP & LOGIN PAGE**



The image shows a dashboard welcome page with a blue background. At the top left is the text "Quizzzy Brainiacs" followed by a hamburger menu icon. At the top right is a "Log out" button with an external link icon. In the center, the text "Hello, kowsi" is displayed in a large font, with the tagline "Inspire, teach, and lead" below it in a smaller font.

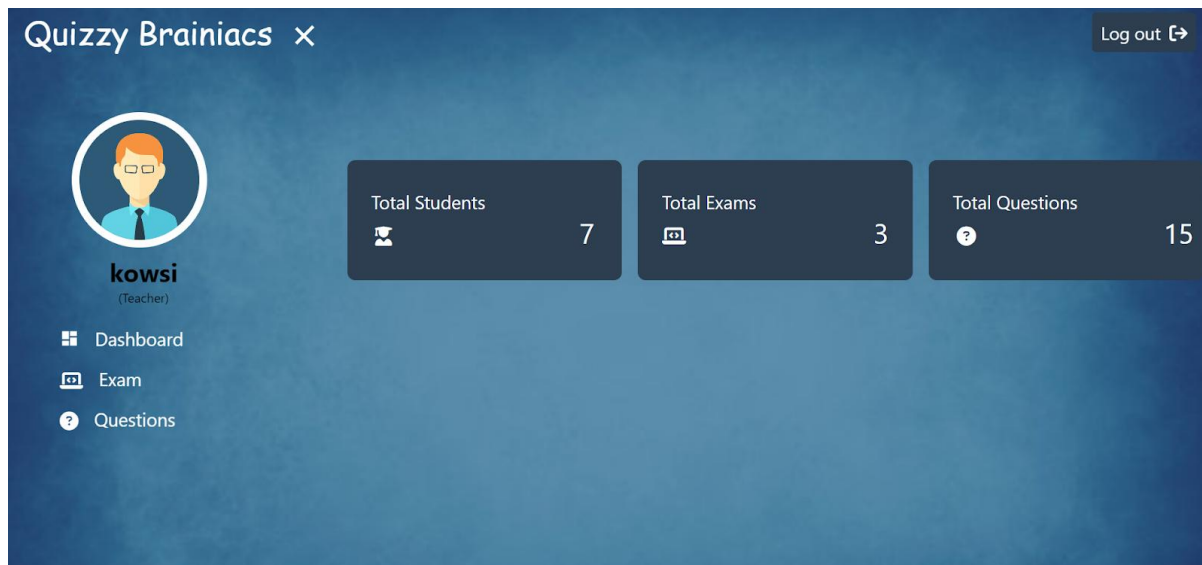
Quizzzy Brainiacs ☰

Log out ➞

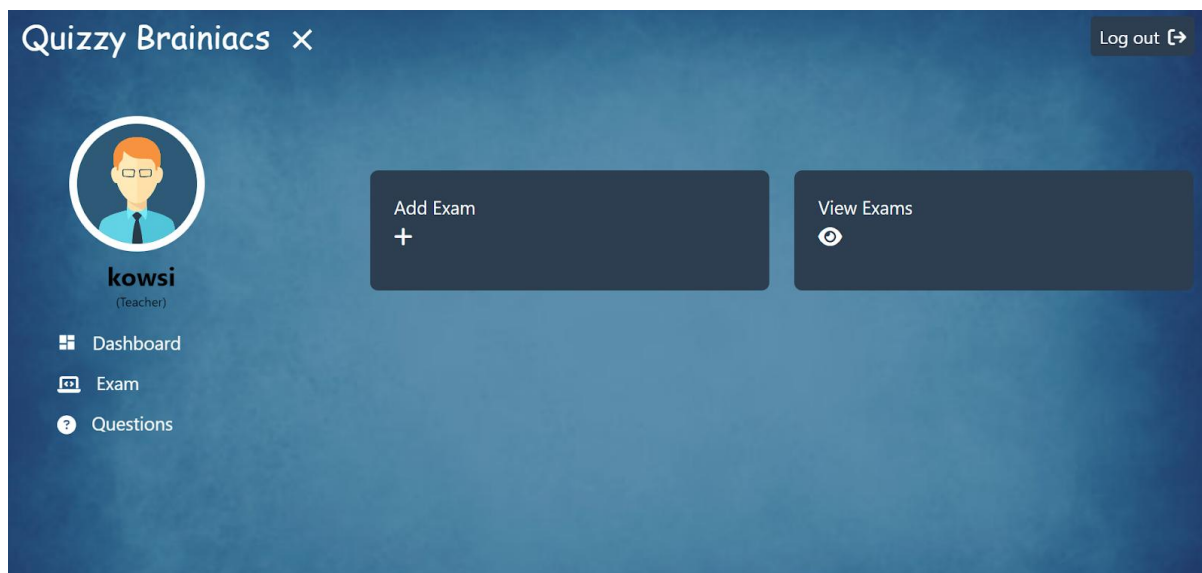
**Hello, kowsi**

Inspire, teach, and lead

**Fig: 12.4 TEACHER DASHBOARD WELCOME PAGE**



**Fig: 12.5 TEACHER DASHBOARD PAGE**




**Fig: 12.6 EXAM PAGE**



Quizzzy Brainiacs ×

Log out ↗



**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions

## ADD EXAM

Exam Name

Total Questions


Total Marks

ADD

**Fig: 12.7 ADD EXAM PAGE**









Quizzzy Brainiacs ×

Log out ↗



**kowski**  
(Teacher)


- Dashboard
- Exam
- Questions

Exams				
Exam Name	Total Questions	Total Marks	Edit	Delete
Express.js	5	5		
React.js	5	5		
Node.js	5	5		
HTML	5	5		

**Fig: 12.8 VIEW EXAM PAGE**

Quizzzy Brainiacs ×

Log out ↗



**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions

## Update Exam

**Exam name**  
Express.js

**Total Question**  
5


**Total Marks**  
5

Update

**Fig: 12.9 UPDATE EXAM PAGE**

Quizzzy Brainiacs ×

Log out ↗



**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions


Add Question  
+

View Questions  
👁

**Fig: 12.10 QUESTION PAGE**

Quizzy Brainiacs
×

Log out



**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions

## ADD QUESTION

**Exam**

Node.js

**Question**

How modules in Node.js can be connected from one component to another ?

**Marks**

1

**Option 1**

Expose

**Option 2**

Module


**Option 3**

Exports

**Fig: 12.11 ADD QUESTION PAGE**

Quizzy Brainiacs
×

Log out




**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions

Select Course To View Questions			
Course Name	Total Questions	Total Marks	View Questions
Express.js	5	5	
React.js	5	5	
Node.js	5	5	

**Fig: 12.12 SELECT COURSE TO VIEW QUESTION PAGE**

Quizzy Brainiacs
×
Log out



**kowski**  
(Teacher)


- Dashboard
- Exam
- Questions

### Questions

Question	Marks	Edit	Delete
Which of following command starts a REPL session?	1		
Which of the following module is required for exception handling in Node?	1		
When a JavaScript function is called in Node.js, where is a new frame placed?	1		
Which of the following is a core module in Node?	1		
What is the default scope in the Node.js application?	1		
How modules in Node.js can be connected from one component to another ?	1		

**Fig: 12.13 VIEW QUESTION PAGE**

Quizzy Brainiacs
×
Log out



**kowski**  
(Teacher)

- Dashboard
- Exam
- Questions

## Update Question

**Exam**

Node.js

**Question**

How modules in Node.js can be connected from one component to another ?

**Marks**

1

**Option 1**

Expose

**Option 2**

Module

**Option 3**

Exports

**Fig: 12.14 UPDATE QUESTION PAGE**

# Hello, Student

Welcome to Quizzy Brainiacs

You can access various features after login.

Create Your Account >>

Log in ↗

**Fig: 12.15 STUDENT PAGE**

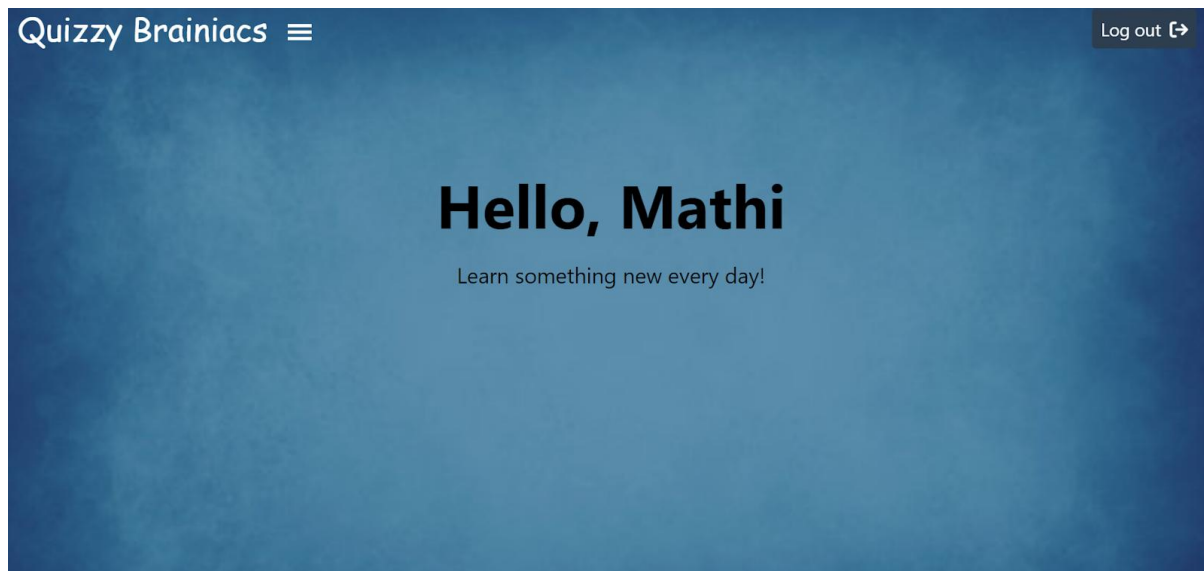
## Student Login

mathi@gmail.com
*****
Log in

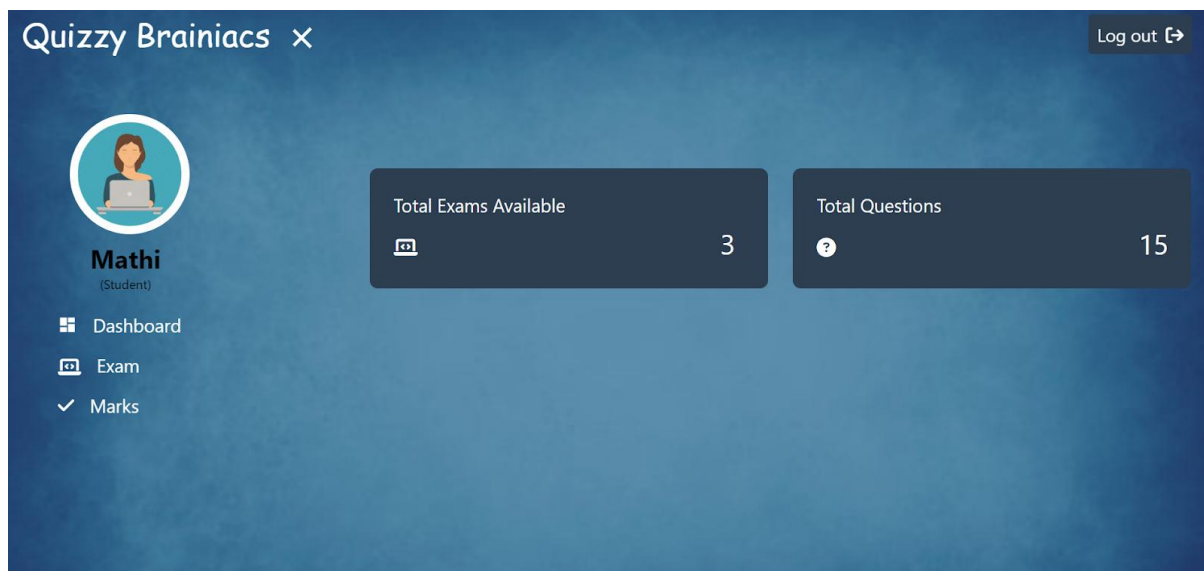
## Student Signup

Mathi
mathi@gmail.com
*****
Sign up

**Fig: 12.16 STUDENT SIGN UP & LOGIN PAGE**

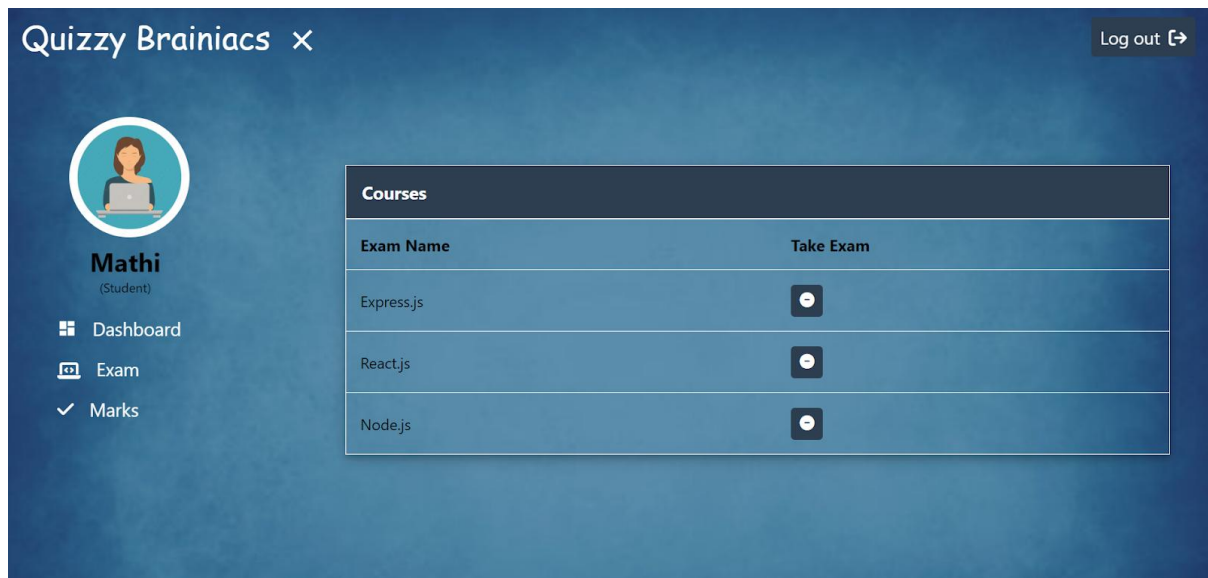


**Fig: 12.17 STUDENT DASHBOARD WELCOME PAGE**

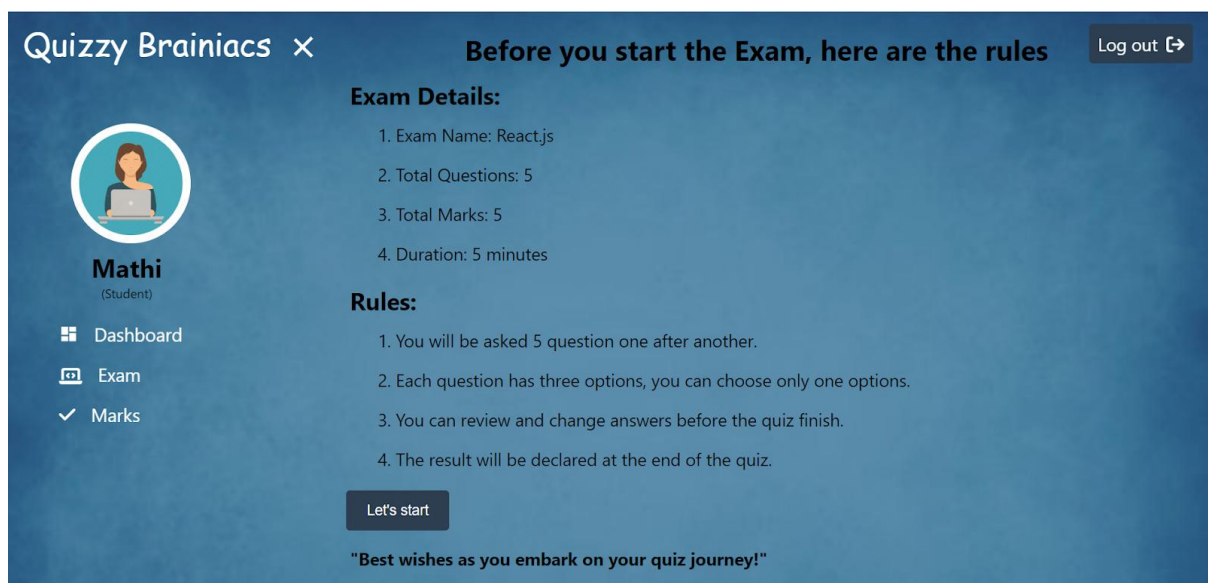


**Fig: 12.18 STUDENT DASHBOARD PAGE**

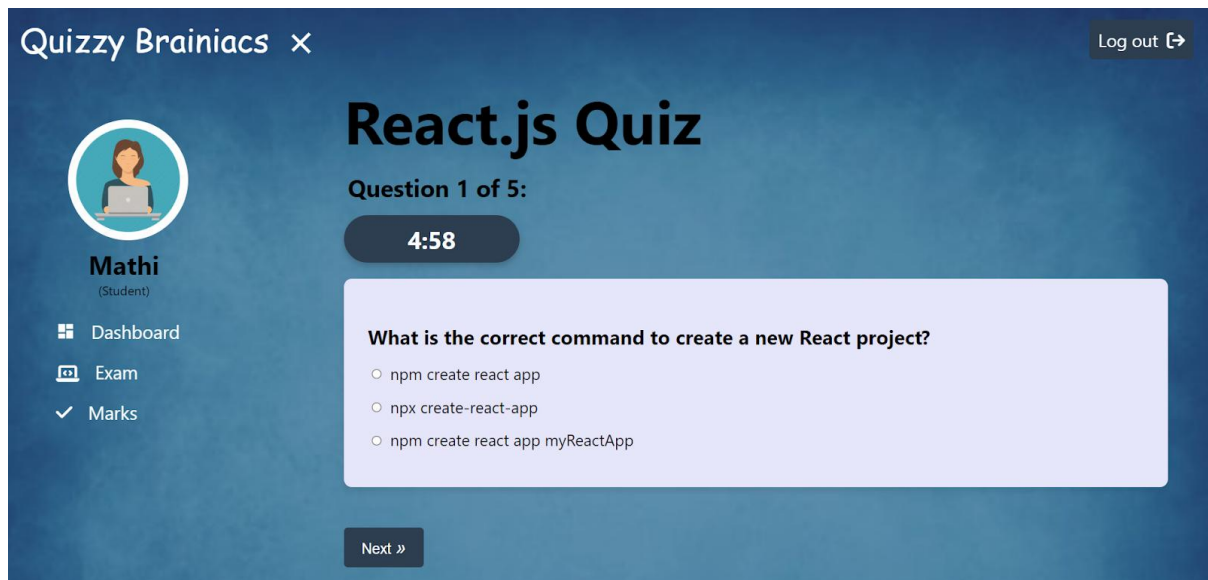




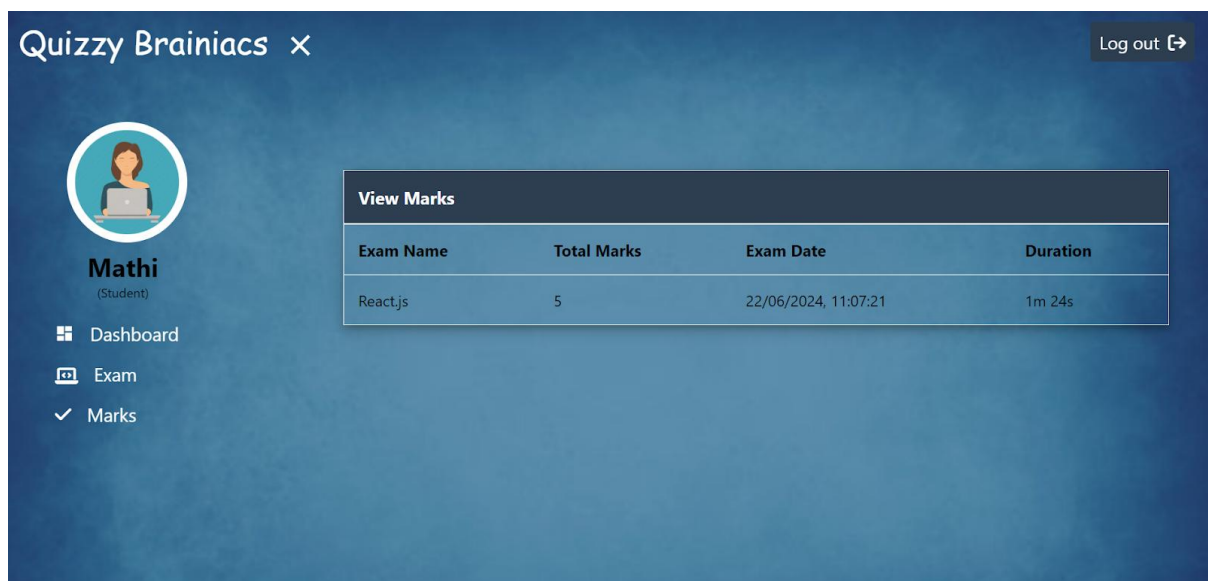
**Fig: 12.19 TAKE EXAM PAGE**



**Fig: 12.20 EXAM RULES PAGE**



**Fig: 12.21 QUESTION LIST PAGE**



**Fig: 12.22 MARK LIST PAGE**



## CHAPTER 13

### REFERENCES

- [1] R. Arora, "Full Stack Development with MERN: Build your own Quiz App," *Packt Publishing*, 2020.
- [2] M. Perez, "MERN Stack Web Development: Quiz Application Tutorial," *Medium*, 2019. [Online]. Available: <https://medium.com>
- [3] J. Doe, "Building a Quiz Application with React and Node.js," *Dev.to*, 2021. [Online]. Available: <https://dev.to>
- [4] A. Kumar, "Implementing Authentication in MERN Stack Applications," *Auth0 Blog*, 2020. [Online]. Available: <https://auth0.com/blog>
- [5] T. Smith, "Using MongoDB for Data Storage in a Quiz Application," *MongoDB Blog*, 2019. [Online]. Available: <https://www.mongodb.com/blog>
- [6] K. Johnson, "Testing React Applications with Jest and React Testing Library," *TestingLibrary*, 2020. [Online]. Available: <https://testing-library.com/docs/react-testing-library/intro/>
- [7] S. Patel, "Integrating Redux for State Management in React Applications," *ReduxDocumentation*, 2021. [Online]. Available: <https://redux.js.org/introduction/getting-started>
- [8] B. Lee, "Deploying MERN Applications to Cloud Platforms," *AWS DeveloperBlog*, 2020. [Online]. Available: <https://aws.amazon.com/blogs/developer>
- [9] C. Davis, "User Authentication and Authorization in MERN Applications," *OktaDeveloper Blog*, 2021. [Online]. Available: <https://developer.okta.com/blog>
- [10] L. Anderson, "Creating RESTful APIs with Node.js and Express," 2020. [Online]. Available: <https://expressjs.com/en/starter/basic-routing.html>