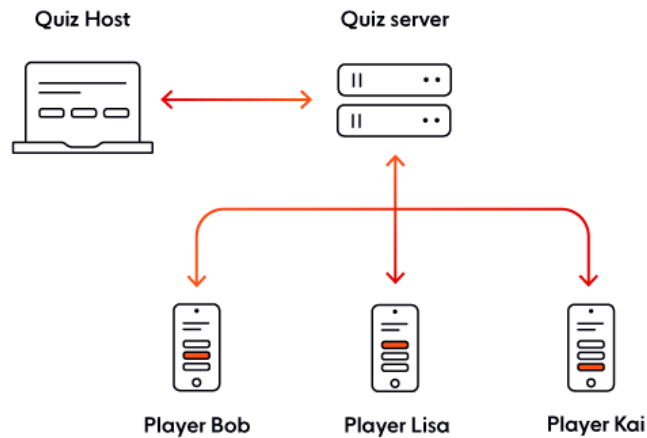


QuizDeck - Technical Design Document

Table of Contents

1. **Introduction**
 2. **System Overview**
 3. **Architecture**
 - High-Level Architecture
 - Component Interactions
 4. **Backend Design**
 - Technologies Used
 - Project Structure
 - API Design
 - Database Schema
 - Middlewares
 5. **Frontend Design**
 - Technologies Used
 - Project Structure
 - Routing
 - State Management
 - Key Components
 6. **Security Considerations**
 7. **Deployment Plan**
 8. **Unit Testing**
 9. **Future Enhancements**
 10. **Conclusion**
-



1. Introduction

QuizDeck is a web-based platform that enables users to create, manage, and participate in quizzes. The application offers real-time interactions, user authentication, and a responsive user interface to enhance the quiz-taking experience. It ensures scalability and ease of use and is designed to handle high user traffic and complex interactions.

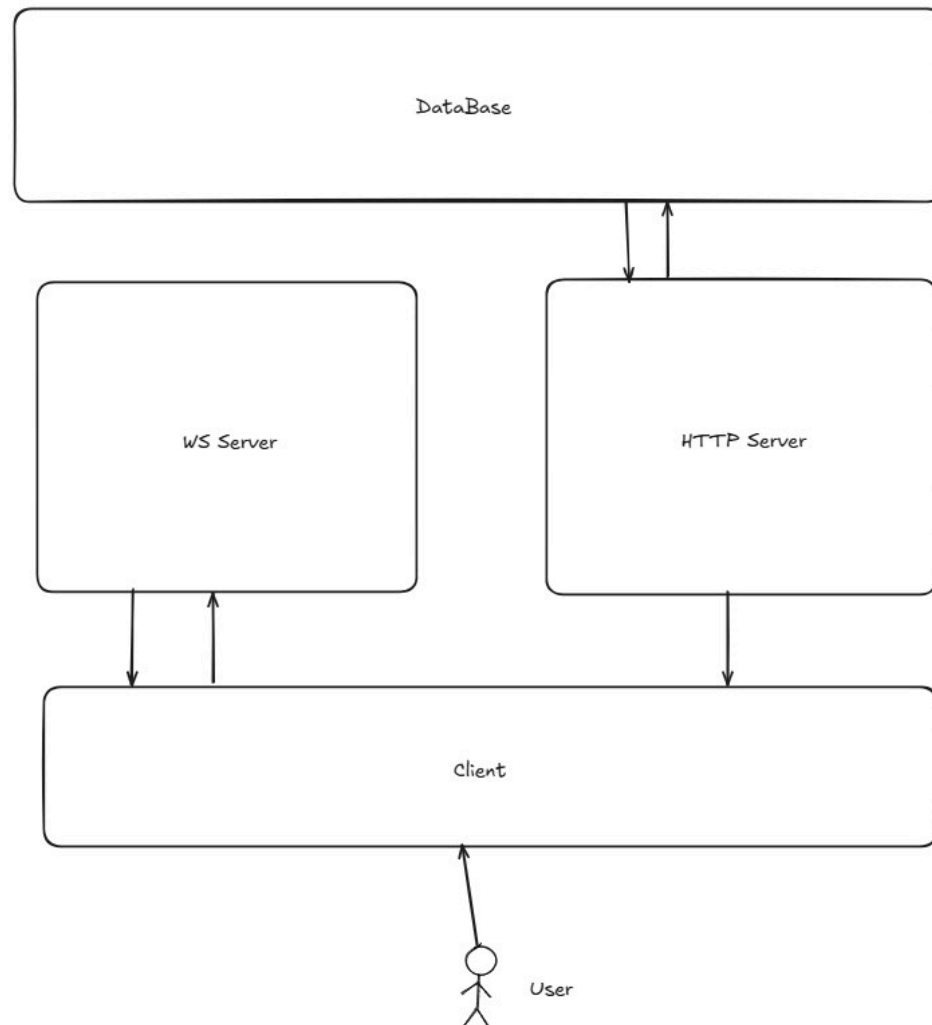
2. System Overview

QuizDeck utilises the MERN stack (MongoDB, Express.js, React.js, Node.js) to deliver a full-stack JavaScript solution. It incorporates modern web development practices to provide a seamless, engaging user experience across devices.

3. Architecture

High-Level Architecture

The application follows a three-tier architecture:



- **Frontend:** Built with React.js, the frontend handles user interactions and presents data dynamically.
- **Backend:** Express.js and Node.js manage server-side logic, APIs, authentication, and business logic.
- **Database:** MongoDB stores user data, quiz information, and results securely and efficiently.

Component Interactions

1. **User requests** are initiated from the frontend and sent to the backend via RESTful APIs or WebSocket connections.
2. **Backend processing** includes authentication, data validation, and database operations.
3. **Responses** are sent back to the frontend for presentation or further interactions.

4. Backend Design

Technologies Used

- **Node.js**: JavaScript runtime environment.
- **Express.js**: Web application framework for building robust APIs.
- **MongoDB**: NoSQL database for data storage.
- **Mongoose**: ODM for MongoDB.
- **Socket.IO**: Real-time communication.
- **JWT**: JSON Web Tokens for secure authentication.
- **bcrypt**: For hashing passwords.

Project Structure

```
backend/
├── controllers/
│   ├── user.controller.js
│   └── quizController.js
├── middlewares/
│   └── authMiddleware.js
├── models/
│   ├── User.js
│   └── Quiz.js
├── routes/
│   ├── user.js
│   └── quiz.js
├── services/
│   ├── room.service.js
│   └── socket.service.js
└── index.js      # Entry point
```

API Design

WebSocket Events

Client-Sent Events:

- **joinRoom**: Allows a user to join a specific quiz room. Payload: `{ roomId: string, userId: string }`
- **submitAnswer**: Submits an answer for the current quiz question. Payload: `{ questionId: string, answer: string }`

- **leaveRoom**: Notifies the server that a user has left a room. Payload: `{ roomId: string, userId: string }`

Server-Sent Events:

- **roomJoined**: Acknowledges that a user has successfully joined a room. Payload: `{ roomId: string, userId: string }`
- **newQuestion**: Sends the next quiz question to all users in the room. Payload: `{ questionId: string, questionText: string, options: string[] }`
- **answerResult**: Provides feedback on whether the submitted answer is correct. Payload: `{ isCorrect: boolean, correctAnswer: string }`
- **leaderboardUpdate**: Sends updated leaderboard rankings for the room. Payload: `{ leaderboard: Array<{ userId: string, score: number }> }`

Authentication APIs

- **POST /user/signup**: Register a new user.
- **POST /user/signin**: Authenticate user and issue JWT.

Quiz APIs

- **POST /quiz/create**: Create a new quiz.
- **GET /quiz/user**: Retrieve all quizzes.
- **GET /quiz/:id**: Retrieve a specific quiz.
- **PUT /quiz/:id**: Update a specific quiz.
- **DELETE /quiz/:id**: Delete a specific quiz.

Database Schema

User Model

```
{
  "username": "string",
  "email": "string",
  "password": "string",
  "quizzes": ["ObjectId"]
}
```

Quiz Model

```

{
  "title": "string",
  "description": "string",
  "questions": [
    {
      "questionText": "string",
      "options": ["string"],
      "correctAnswer": "string"
    }
  ],
  "author": "ObjectId",
  "createdAt": "Date",
  "updatedAt": "Date"
}

```

Middlewares

- **Authentication Middleware:** Verifies JWT and authorizes access to protected routes.-
-

5. Frontend Design

Technologies Used

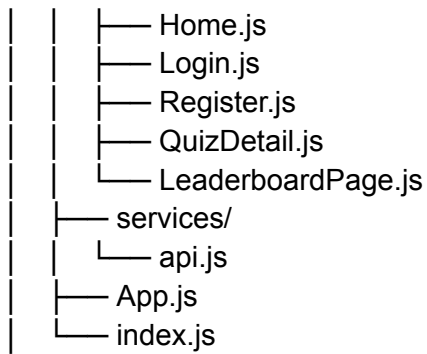
- **React.js:** JavaScript library for building user interfaces.
- **React Router DOM:** For client-side routing.
- **Axios:** For making HTTP requests.
- **Bootstrap:** For responsive design and styling.
- **Tailwind CSS:** Utility-first CSS framework for scalable styling.

Project Structure

```

client/
├── public/
├── src/
│   ├── components/
│   │   ├── Navbar.js
│   │   ├── QuizForm.js
│   │   ├── Leaderboard.js
│   │   └── QuizList.js
│   └── pages/

```



Routing

- `/`: Home Page
- `/login`: User Login
- `/register`: User Registration
- `/quizzes`: List of Quizzes
- `/quizzes/:id`: Quiz Details
- `/leaderboard`: View Leaderboard

State Management

- **React Context API**: Manages global state, including user authentication status.
- **Local Component State**: Manages form inputs and local UI states.
- **Socket.IO**: Handles real-time quiz updates.

Key Components

- **Navbar**: Navigation bar with links to main sections.
- **QuizForm**: Form for creating and editing quizzes.
- **QuizList**: Displays a list of available quizzes.
- **QuizDetail**: Shows details of a selected quiz.
- **Leaderboard**: Displays top scores for a quiz.

6. Security Considerations

Authentication

- **JWT**: Secure token-based authentication.
- **Password Hashing**: Passwords are hashed using bcrypt before storage.

Authorization

- **Role-Based Access Control:** Ensures only authorized users can create, edit, or delete quizzes.

Data Validation

- **Input Sanitization:** Prevents injection attacks by validating and sanitizing user inputs.
-

7. Deployment Plan

Environment Variables

- **Backend:**
 - `MONGO_URI`: MongoDB connection string.
 - `JWT_SECRET`: Secret key for signing JWTs.
- **Frontend:**
 - `REACT_APP_API_URL`: Base URL for API endpoints.

Deployment Steps

1. **Backend:**
 - Deploy to a platform like Heroku or AWS.
 - Set environment variables.
 - Ensure MongoDB database is accessible.
 2. **Frontend:**
 - Deploy to a platform like Netlify or Vercel.
 - Configure environment variables.
 - Ensure the frontend is connected to the backend API.
 3. **Testing:**
 - Conduct end-to-end testing to ensure seamless interactions.
-

8. Unit Testing

- **Backend:**
 - Use Jest for testing API endpoints.
 - Mock database operations using tools like MongoMemoryServer.
- **Frontend:**
 - Use React Testing Library for component tests.

- Write tests for state management and API interactions.
-

9. Future Enhancements

- **Real-Time Features:** Implement live quiz sessions using WebSockets.
 - **User Profiles:** Allow users to view and edit their profiles.
 - **Quiz Analytics:** Provide detailed performance analytics for users and admins.
 - **Gamification:** Add achievements and rewards to increase engagement.
 - **Mobile Optimization:** Enhance the UI for mobile devices.
-

10. Conclusion

QuizDeck provides an engaging platform for creating and participating in quizzes. By leveraging the MERN stack, the project ensures scalability, maintainability, and responsiveness. Additional features such as real-time interactions, gamification, and analytics will enhance the platform's appeal and usability.