Hackathon Problem Statement: "1V1 MCQ Battle"

Overview

Participants will create a real-time "1V1 MCQ Battle" web application where users can challenge each other in a quiz-based game. The application will consist of a backend built with Django or any backend framework of your choice to manage the game logic, user interactions, and data storage, and a React frontend to handle the user interface and real-time interactions using Pusher for websockets.

Objective

The goal is to build a fully functional web application where users can sign up, join a lobby, challenge other players to a multiple-choice question (MCQ) battle, and answer questions in real-time. The application will track scores and declare winners at the end of each game.

Tools and Technologies

- **Backend**: Django / Node.js, Express.js for REST APIs
- **Database**: MongoDB or any preferred SQL/NoSQL database
- **Real-time Communication**: Pusher for handling real-time functionalities
- **Frontend**: React.js
- **Styling**: Antd, CSS, Bootstrap, or any preferred CSS framework

Tutorials

Web Development Essentials | AlgoZenith (maang.in)
Django documentation | Django documentation | Django (djangoproject.com)
Introduction to React Js + Installation | Complete React Course in Hindi #1 (youtube.com)
Pusher Channels Docs | JavaScript Quick Start Guide
Build a Fullstack CHAT App using MERN (mongo, express, react, node) (youtube.com)

Step 1: Backend - Setting up User Entity, Authentication, and Creating MCQ Entities and CRUD APIs

Objective

Develop a RESTful API to manage user entities and MCQs, including authentication and CRUD operations for the MCQ entities.

Tasks

1. **Set Up Project and Database Connection**
    - Initialize a Django / Node.js project and install necessary packages.
    - Establish a connection to the database.
2. **User Entity and Authentication**
    - **Model Creation**: Define a `User` schema with attributes such as username, email, password, and any additional relevant information.
    - **Password Handling**: Implement password hashing using libraries like `bcrypt` to ensure security.
    - **Signup API**: Create an endpoint to register new users. This API should handle user data validation, password hashing, and storing new user records in the database.
    - **Login API**: Develop an endpoint for user login that validates user credentials and returns a token (consider using JWT for session management).
    - **Authentication Middleware**: Create middleware to authenticate API requests, ensuring that certain endpoints are accessible only to logged-in users.
3. **MCQ Entity Management**
    - **Schema Definition**: Create a DB schema for MCQs, which includes fields for the question, multiple choice options, the correct answer, and optional metadata such as difficulty level.
    - **CRUD APIs for MCQs**:
        - **Create**: API to add new questions, ensuring only authenticated users can add questions.
        - **Read**: API to fetch questions.
        - **Update and Delete**: APIs to modify and remove existing questions, secured to prevent unauthorized access.

Considerations

- **API Security**: Implement rate limiting and other security best practices to protect the APIs.
- **Data Validation**: Ensure robust server-side validation for all inputs to protect against invalid or malicious data.
- **Error Handling**: Implement comprehensive error handling to manage and return appropriate error responses for various failure conditions.

Step 2: Frontend Development - User Interface for Signup, Login, and MCQ Management

Objective

Build a responsive and user-friendly web interface using React for user signup, login, and managing MCQ entities.

Tasks

1. **Setup React Application**
    - **Create React App**: Start by creating a new React project using Create React App or any other starter template you prefer.
    - **Project Structure**: Organize the project into a sensible directory structure, with separate folders for components, services (for API calls), and utilities.
2. **Implement User Authentication UI**
    - **Signup Component**: Create a form that collects username, email, and password, and calls the signup API on submission. Implement client-side validation to ensure data integrity before submission.
    - **Login Component**: Develop a login form that takes a username and password. On submission, authenticate with the backend and handle session management using context or Redux to store the received authentication token.
    - **Navigation Bar**: Add a navigation component that provides links to different parts of the application and displays user login status with options to log out or navigate to the MCQ management page, depending on whether the user is logged in or not.
3. **Develop MCQ Management Interface**
    - **List MCQs Component**: Create a component to display a list of all MCQs retrieved from the backend. This component should allow authenticated users to view, edit, or delete questions.
    - **Add/Edit MCQ Form**: Build a form to submit new MCQs or update existing ones. This form should include fields for the question, options, and the correct answer, and should handle both the creation and updating of MCQs depending on the context (new or edit).
    - **Delete Functionality**: Implement a delete button in the MCQ list that allows users to delete questions with a confirmation prompt before performing the deletion.
4. **Connect Frontend to Backend**
    - **API Services**: Create a set of services using Axios or Fetch API to handle all backend communications, such as getting, posting, updating, and deleting MCQs.
    - **State Management**: Use React context, Redux, or another state management library to manage the state across the application, especially handling user authentication status and the MCQ data.
5. **Styling and Responsiveness**

    - **CSS/Styling**: Use CSS, Sass, or any CSS-in-JS library like styled-components to style the components. Ensure the interface is intuitive and visually appealing.
    - **Responsive Design**: Make sure the UI is responsive and provides a consistent experience across different devices and screen sizes.

Considerations

- **User Experience**: Focus on creating a seamless user experience with clear navigation and immediate feedback on user actions (like displaying loading indicators during API requests).
- **Error Handling**: Provide user-friendly error messages for various errors, such as network issues, login failures, or unauthorized actions.

Step 3: Implementing Game Mechanics and Lobby System

Objective

Develop the core functionality of the "1V1 MCQ Battle" including the game creation, lobby system where created games are displayed, and the game logic itself. This involves handling game invitations, participant interactions, and the progression of the game based on time and the correctness of answers.

Tasks

1. **Game and Lobby Models**
    - **Create Game Model**: Develop a model on the backend to represent a game session. It should include fields such as game ID, owner (creator of the game), game status (waiting, active, completed), and participants.
    - **Game Creation API**: Implement an API to create a new game. The game should initially be in the 'waiting' status and appear in the lobby as an open session that others can join.
    - **List Games API**: Create an API to list all games currently in the 'waiting' status in the lobby.
2. **Lobby Interface**
    - **Display Available Games**: In the React frontend, develop a component that fetches and displays a list of available games from the backend. Each game should show basic details like the game ID, owner, and a button to request to join.
    - **Request to Join Game**: Implement functionality for users to send a request to join a selected game. This could involve a WebSocket or another real-time method to notify the game owner.
3. **Game Invitation Handling**

    - **Acceptance of Game Requests**: Allow the game owner to accept join requests from other users. Once a request is accepted, update the game status to 'active' and notify both players.
    - **Pusher for Real-time Updates**: Use Pusher to manage real-time interactions in the lobby and during the game. This includes notifications for game requests, game start, and real-time game data exchange.
4. **Gameplay Mechanics**
    - **MCQ Delivery System**: Develop a system to serve MCQs to the participants. Questions should be sent one at a time and continue until either 10 minutes have elapsed or a participant makes three incorrect answers.
    - **Answer Validation and Scoring**: Implement logic to validate submitted answers against the correct answers stored in the database. Keep a running score, and track the number of incorrect answers for each participant.
    - **Game Termination Conditions**: End the game if the 10-minute timer expires or if a participant makes three incorrect answers. Determine and declare the winner based on the scores.
5. **Ending the Game and Declaring a Winner**
    - **Determine Winner**: At the end of the game, compare scores and declare the winner. If a player is disqualified due to incorrect answers, the other player automatically wins.
    - **Result API**: Implement an API to send the final results to both players and update the game status to 'completed'.
    - **Display Results**: Show the game results on the frontend, including the winner and the final scores.

Considerations

- **Efficiency and Scalability**: Ensure the game logic and real-time interactions are efficient and can scale to handle multiple simultaneous users and games.
- **User Experience**: Focus on creating an engaging and interactive user experience, with smooth transitions and immediate feedback during the game.
- **Security**: Handle all data exchanges securely, particularly the transmission of user inputs and game results.
- **Robustness**: Implement error handling and recovery mechanisms to deal with potential issues like network failures or unexpected user disconnections.

Step 4: Frontend Development for Game System and API Integration

Objective

Complete the frontend development for the game system in the "1V1 MCQ Battle" application, focusing on real-time gameplay interaction, and integrate the backend APIs to ensure a seamless user experience for participants engaging in the MCQ battles.

Tasks

1. **Game Lobby UI**
    - **Create Lobby Component**: Develop a React component for the game lobby that lists all available games. This component should refresh in real-time using Pusher to show new games as they are created.
    - **Join Game Functionality**: Implement a button or link for each listed game that allows users to send a join request to the game owner. Handle these requests in real-time to provide immediate feedback to the user about the request status.
2. **Gameplay UI**
    - **Game Component**: Design and implement a React component for the actual gameplay. This component should display the MCQs, multiple-choice options, a timer, and live score updates.
    - **Real-Time Question Handling**: Integrate with Pusher to receive questions in real-time. Ensure that the questions are displayed one at a time and that users can submit their answers through the interface.
    - **Timer Implementation**: Add a countdown timer for the 10-minute game duration. Display a warning as the end of the game approaches.
3. **Integrate Backend APIs**
    - **Start Game API**: Once a join request is accepted, use the API to transition the game status from 'waiting' to 'active'. This should trigger the start of the game and the countdown timer.
    - **Submit Answers API**: Develop functionality to submit answers back to the server for validation. Use async operations to handle responses and update the game state accordingly.
    - **Results API**: At the end of the game, fetch results from the backend and display them to both participants, showing the winner and final scores.
4. **User Interactions and Real-Time Updates**
    - **Websockets/Pusher Integration**: Use Pusher for handling all real-time aspects of the game, including receiving new questions, submitting answers, and getting score updates.
    - **Handle Disconnections and Reconnections**: Implement logic to handle cases where a user might disconnect and try to reconnect to the game.
5. **Styling and User Experience**
    - **Responsive Design**: Ensure that the game UI is responsive and provides an engaging user experience across different devices and screen sizes.
    - **Interactive Elements**: Use animations and transitions to make the gameplay more engaging. For instance, animate the transitions between questions and provide visual feedback for correct and incorrect answers.
    - **Feedback Mechanisms**: Provide immediate visual feedback for user actions, such as submitting an answer or receiving a score update.

Considerations

- **Performance**: Optimize the frontend for performance, especially during real-time interactions to minimize latency.
- **Testing**: Thoroughly test the frontend with multiple users to ensure the system handles concurrent games gracefully.
- **Security and Validation**: While the backend will handle most security and validation, ensure that the frontend also validates user inputs to prevent malformed data from affecting the game logic.

Bonus Challenges

- Leaderboard system to show top scores.
- Time limits for answering each question.
- Add animations for better user experience.
- Implement unit tests for both backend and frontend.

Evaluation Criteria

- Functionality: Completeness of features as per the problem statement.
- Code Quality: Clarity, maintainability, and use of best practices.
- Design and User Experience: How intuitive and responsive the user interface is.
- Creativity: Originality of the approach taken.
- Bonus Features: Implementation of additional features beyond the basic requirements.

Submission Requirements

- Source code must be submitted via GitHub.
- A README file with setup instructions and a brief explanation of the project structure and technologies used.
- A demo video showcasing the functionality of the application.

Participants are encouraged to plan their implementation strategy and manage their time effectively to create a polished and functional application. Good luck!

Submission Deadline

July 05th, 2024