Online Judge CodeCraft

Problem Statement:

An Online Judge is a platform where coding enthusiasts and competitive programmers can practice and improve their coding skills. Users can solve coding problems, submit their solutions, and get feedback on how correct and efficient their code is. The platform also has a leaderboard to encourage competition and makes sure that code submissions are run safely and efficiently. Some popular online judges are LeetCode, HackerRank, and Codeforces.

Features:

- **Problem Browsing and Viewing:** Users can browse through a list of coding problems and view detailed descriptions, including the problem statement, name, code, difficulty level, and example test cases.
- **User Authentication:** Secure user login and signup functionality ensure safe access to the platform, with encrypted storage of user data.
- **Code Submission and Evaluation:** Users can submit their code for problem solutions, which undergo real-time evaluation in a secure sandbox environment using Docker. Verdicts (e.g., "Accepted", "Wrong Answer") are provided based on predefined test cases.
- **Submission History and Logs:** Accessible logs of recent submissions allow users to track their progress, displaying details such as verdicts, time of submit and programming language used.
- **Profile Management:** Users can view their profiles, where they can find a clear overview of their solved problems.
- **Leaderboard:** A dynamic leaderboard showcases top performers based on successful submissions, fostering a competitive environment and motivating users to excel.

High Level Design:

Database Design

Collection 1: Problems

Document Structure:

- `title` (String, CharField): The title of the problem. (Required, Unique)
- `difficulty` (String, CharField): The difficulty level of the problem. (Required, Enum: "Easy", "Medium", "Hard")
- `codingScore` (Number, IntegerField): The score assigned to the problem. (Required)
- `tags` (Array of Strings, CharField): Tags associated with the problem. (Required)
- `description` (String, CharField): The problem description. (Required)
- `inputFormat` (String, CharField): The format for the input. (Required)

- `outputFormat` (String, CharField): The format for the output. (Required)
- `constraints` (String, CharField): Constraints for the problem. (Required)
- `sampleInput` (String, CharField): An example test case input. (Required)
- `sampleOutput` (String, CharField): An example test case output. (Required)
- `hiddenTestcases` (Array of Objects, ObjectField): Hidden test cases with input and output fields. (Default: [])
- `explanation` (String, CharField, Optional): Explanation for the problem.

Collection 2: Submissions

Document Structure:

- `username` (String, CharField): The username of the person who submitted the solution. (Required)
- `title` (String, CharField): The title of the associated problem. (Required)
- `status` (Object, ObjectField): The status of the submission containing:
    - `success` (Boolean, BooleanField): The result of the solution. (Required)
    - `pass` (Number, IntegerField): The number of test cases passed. (Required)
    - `error` (String, CharField): Error messages if any. (Required)
- `lang` (String, CharField): The programming language of the solution code. (Required)
- `code` (String, CharField): The solution code. (Required)
- `timestamp` (Date, DateTimeField): The submission timestamp. (Default: Current Date and Time)

Collection 3: Users

Document Structure:

- `username` (String, CharField): The unique identifier for the user. (Required, Unique)
- `email` (String, CharField): The user's email address. (Required)
- `roles` (Object, ObjectField): Roles assigned to the user containing:
    - `User` (Number, IntegerField): Role level for a regular user.
    - `Admin` (Number, IntegerField): Role level for an admin.
- `password` (String, CharField): The user's password. (Required)
- `refreshToken` (Array of Strings, CharField): Tokens for refreshing sessions.
- `solved` (Array of Objects, ObjectField): Solved problems with the following fields:
    - `title` (String, CharField): The title of the solved problem. (Required)
    - `difficulty` (String, CharField): The difficulty level of the solved problem. (Required)
    - `codingScore` (Number, IntegerField): The score of the solved problem. (Required)

Web Server Design:

UI Screens

Basic Components

1. **Navbar**
    - **Guest User (no login):**
        - Home
        - Problems
        - Leaderboard
        - Login
        - Register
    - **Normal Users:**
        - Home
        - Problems
        - Submissions
        - Leaderboard
        - Profile
        - Logout
    - **Admin Users:**
        - Home
        - Problems
        - Problem Dashboard
        - Leaderboard
        - Profile
        - Logout
2. **Footer**
    - Available on every page.

Screens

1. **Home Screen**
    - Contains links to all other nav links available based on user type (guest, normal user, admin).
2. **Problem Screen**
    - **Components:**
        - List of problems with the following details:
            - Title (with link to individual problem page)
            - Coding score
            - Difficulty
            - Tags
        - Search bar
        - Filter option
3. **Codespace**

- **Problem Description:**
  - Problem preview with detailed problem description
- **Code Editor:**
  - **Menu Bar:**
    - Language selection dropdown
    - Theme toggle button
    - Code refresh button
  - **Code Editor:**
    - Advanced code editor with features provided by the React Ace libraries.
    - Support for Java, C++, and Python.
  - **Console:**
    - Three tabs:
      - **Input:** For custom input provided by the user.
      - **Output:** Displays output for custom input when code is run.
      - **Verdict:** Shows submission status and number of test cases passed.
  - **Action Buttons:**
    - Console button: Toggles (opens/closes) the console section.
    - Run button: Runs the code on custom input.
    - Submit button: Submits the solution to the problem.

4. **Leaderboard Screen**
   - **Components:**
     - List of users ranked by coding score with the following details:
       - Username (with link to their profile)
       - Rank
       - Total number of problems solved (easy, medium, hard, total)
       - Coding score
5. **Login Page**
   - **Components:**
     - Input fields for username and password
     - Button to log the user into the website as user/admin
6. **Register Page**
   - **Components:**
     - Input fields for:
       - Username
       - Email
       - Password
       - Confirm password
     - Button to create a new user and log the user into the website as a normal user
7. **Submissions Screen**
   - **Components:**

        - List of all submissions made by the user with the following details:
            - Problem title (with link to the problem)
            - Language used
            - Verdict
            - Time of submission
            - Button to view the submitted code
        - Search bar
        - Filter option
8. **Profile Page**
    - **Components:**
        - Username
        - Email ID
        - Coding score
        - List of all solved problems with their links
        - Chart for visualizing the problems solved by level of difficulty
9. **Problem Dashboard (New Problem Page)**
    - Accessible only to admins
    - **Components:**
        - Form to input each attribute of the problem:
            - Title
            - Description
            - Difficulty
            - Coding score
            - Tags
            - Input format
            - Output format
            - Constraints
            - Sample input
            - Sample output
            - Hidden test cases
            - Explanation
        - Preview of the problem being built
        - Create button
10. **Edit Problem Page**
    - Accessible only to admins
    - **Components:**
        - Similar interface to the new problem page
        - Form pre-filled with existing problem data
        - Buttons:
            - Update
            - Delete

Functional Requirements

List Problems

Frontend:

- Create a simple list UI in React that displays the titles of each problem.
- Each problem title should link to its individual problem page.
- **Components:**
    - Navbar
    - Footer
    - Problem List: Display problem title, coding score, difficulty, and tags with a search bar and filter option.

Backend:

- Define an API endpoint in Express.js that handles a GET request to fetch all problems from the database (MongoDB) and return them to the frontend.
- **Endpoint:**
    - `GET /problemset/`

Show Individual Problem

Frontend:

- Design a template in React to display the problem details including title, description, input format, output format, constraints, sample input/output, and a submission box for problem code in text format.
- **Components:**
    - Navbar
    - Footer
    - Problem Description
    - Code Submission Box
    - Language Selection Dropdown
    - Theme Toggle Button
    - Code Refresh Button
    - Console (with tabs for input, output, and verdict)
    - Run Button
    - Submit Button

Backend:

- Define an API endpoint in Express.js to handle a GET request to fetch the problem details from the database and return them to the frontend.
- **Endpoint:**
    - `GET /problemset/:title`

Code Submission

Frontend:

- Include a submit button below the code submission box in the "Show Individual Problem" template.
- **Components:**
    - Submit Button
    - Console Section (for displaying input, output, and verdict)

Backend:

- Define an API endpoint in Express.js to handle a POST request from the frontend. This endpoint should:
    - Retrieve the test cases (input and expected output) for the problem from the database.
    - Evaluate the submission code using a local compiler or interpreter from the backend. Use `child_process` or a similar library to call the system command for compilation or execution.
    - Compare the outputs from the compiler/interpreter to the expected outputs of the test cases.
    - Save the verdict for this submission (e.g., "Accepted," "Wrong Answer," etc.) in the database.
    - Return the verdict and any other relevant data to the frontend.
- **Endpoints:**
    - `POST /submit/:title`
    - `POST /run/`

Show Submissions

Frontend:

- Create a UI in React to display the recent 10 submissions for a specific problem. Each submission should show the submission details, verdict, and any other relevant information.
- **Components:**
    - Navbar
    - Footer
    - Recent Submissions List

Backend:

- Define an API endpoint in Express.js to handle a GET request to fetch the recent 10 submissions for the specified problem from the database and return them to the frontend.
- **Endpoint:**
    - `GET /submissions/`

Leaderboard

Frontend:

- Create a list UI in React to display the verdicts of the last 10 submissions.
- **Components:**
    - Navbar
    - Footer
    - Leaderboard List: Display users ranked by coding score, their profile links, total number of problems solved (easy, medium, hard, total), and coding score.

Backend:

- Define an API endpoint in Express.js to handle a GET request for fetching the solutions along with the verdicts for the last 10 submissions from the database.
- **Endpoint:**
    - `GET /leaderboard/`

Additional Endpoints

User Authentication and Management:

- Register: `POST /register`
- Login: `POST /auth`
- Logout: `GET /logout`
- Refresh Token: `GET /refresh`
- Get User Details: `GET /user/:user`

Problem Management (Admin Only):

- Add Problem: `POST /problemset/`
- Update Problem: `PUT /problemset/:title`
- Delete Problem: `DELETE /problemset/:title`

Evaluation System:

Code Execution and Sandbox Environment

Docker Setup:

- **Containerized Execution:** Use Docker containers to run submitted code, ensuring isolated execution environments.
- **High CPU Machines:** Deploy containers on high CPU machines for efficient computation.
- **Resource Management:**
    - **Sandboxing:** Restrict execution environments to prevent unauthorized access to system resources.

    - **Resource Limits:** Set limits on CPU, memory, and disk usage.
    - **Time Limits:** Enforce execution time limits to prevent long-running processes.

Security Measures:

- **Privilege Restrictions:** Run containers with limited privileges and non-root users.
- **Isolation:** Ensure each container is isolated from others.

Advantages of using MERN:

- **Full Stack Consistency:** MERN provides a consistent JavaScript-based technology stack for both the frontend and backend development. This ensures better coordination among the development team and smoother integration of components.
- **Single Language:** Developers proficient in JavaScript can work on both frontend and backend aspects of the project, reducing the learning curve and facilitating easier code maintenance.
- **React for Interactive UI:** React, being a highly efficient and popular frontend library, allows for the creation of interactive and responsive user interfaces. Its component-based architecture makes it easy to manage complex UI elements.
- **Express.js for Backend:** Express.js, being a minimalist and flexible web application framework for Node.js, allows for quick setup of server-side logic, routing, and middleware integration. It's well-suited for building RESTful APIs, which is crucial for handling requests in an Online Judge system.
- **MongoDB for Database:** MongoDB, a NoSQL database, offers flexibility in handling unstructured data and scalability for managing large volumes of data. Its JSON-like document structure aligns well with JavaScript objects, making data manipulation intuitive for developers.
- **Real-time Data Handling:** Node.js facilitates real-time data processing through event-driven, non-blocking I/O operations, making it suitable for handling concurrent connections and real-time updates, which can be beneficial for features like live submission updates and leaderboard.

Disadvantages of using MERN:

- **Learning Curve:** While JavaScript is a widely known language, mastering the entire MERN stack may require additional learning for developers unfamiliar with certain technologies like React or Node.js.
- **Scalability:** While MongoDB provides scalability, it might not be as robust for complex queries and transactions compared to traditional relational databases. This could potentially lead to performance issues as the system scales.
- **Security Concerns:** Node.js and MongoDB have their own security considerations. Node.js is susceptible to certain types of attacks like DDoS due to its single-threaded nature, and MongoDB requires careful configuration to prevent common security vulnerabilities like injection attacks.

- **Community and Ecosystem:** While the MERN stack has a large and active community, the ecosystem might not be as mature or extensive as some other tech stacks. This could result in fewer available libraries, tools, or resources for specific requirements of the project.
- **Complexity for Simple Applications:** For simpler applications, the MERN stack might introduce unnecessary complexity. Using a full-fledged frontend library like React for a basic CRUD application, for example, might be overkill.
- **Scalability of Node.js:** While Node.js is highly scalable for handling I/O-bound tasks, it might not be as efficient for CPU-bound operations. This could become a bottleneck when executing and evaluating submitted code, especially in scenarios with heavy concurrent user traffic.