Online Judge

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, execution time, memory usage, and programming language used.
- **Profile Management:** Users can manage their profiles, including updating personal information, changing passwords, and viewing submission statistics. Profile customization enhances user experience and engagement.
- **Leaderboard:** A dynamic leaderboard showcases top performers based on successful submissions and problem-solving efficiency, fostering a competitive environment and motivating users to excel.

High Level Design:

1. Database Design

Collection 1: Problems

Document Structure:

- `statement`: string (CharField) - The problem statement.
- `name`: string (CharField) - The name of the problem.
- `code`: string (CharField) - A unique code for the problem.
- `difficulty`: string (CharField, optional) - The difficulty level of the problem.
- `test_case_input`: (CharField) - An example test case input.
- `test_case_output`: (CharField) - An example test case output.

Collection 2: Solutions

Document Structure:

- `problem`: reference to the problem document (Foreign Key) - The associated problem.
- `verdict`: string (CharField) - The result of the solution (e.g., "Accepted", "Wrong Answer").
- `submitted_at`: date and time (Auto DateTime Field) - The submission timestamp.
- `time`: int (IntegerField) - The running time of the solution code (in milliseconds).
- `memory`: int (IntegerField) - The memory usage of the solution code (in kilobytes).
- `lang`: string (CharField) - The programming language of the solution code.

Collection 3: Test Cases

Document Structure:

- `input`: string (CharField) - The input for the test case.
- `output`: string (CharField) - The expected output for the test case.
- `problem`: reference to the problem document (Foreign Key) - The associated problem.

Collection 4: Login/Signup

Document Structure:

- `UserId`: string (CharField) - The unique identifier for the user.
- `Password`: string (CharField) - The user's password.
- `Email`: string (CharField) - The user's email address.
- `DOB`: date (DateField) - The user's date of birth.
- `FullName`: string (CharField) - The user's full name.

2. Web Server Design

UI Screens

1. **Home Screen**
    - **Components:**
        - Problem List
        - Login/Signup
2. **Specific Problem Screen**
    - **Components:**
        - Language Selection
        - File Selection
        - Coding Arena
        - Verdict / Submission Log
3. **LeaderBoard Screen (Optional)**
    - **Components:**
        - List of Top Performers
4. **Show Submissions Screen**
    - **Components:**
        - List of Recent Submissions for a Specific Problem

Functional Requirements

1. **List Problems**
    - **Frontend:**
        - Create a simple list UI in React that displays the names of each problem.
        - Each problem name should link to its individual problem page.
    - **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.
2. **Show Individual Problem**
    - **Frontend:**
        - Design a template in React to display the problem name, statement, and a submission box for problem code in text format.
    - **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.
3. **Code Submission**
    - **Frontend:**
        - Include a submit button below the code submission box in the "Show Individual Problem" template.
    - **Backend:**

        - Define an API endpoint in Express.js to handle a POST request from the frontend. This endpoint should execute the following steps:
            1. Retrieve the test cases (input and expected output) for the problem from the database.
            2. 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.
            3. Compare the outputs from the compiler/interpreter to the expected outputs of the test cases.
            4. Save the verdict for this submission (e.g., "Accepted," "Wrong Answer," etc.) in the database.
            5. Return the verdict and any other relevant data to the frontend.

4. **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.
    - **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.

5. **Leaderboard**

    - **Frontend:**
        - Create a list UI in React to display the verdicts of the last 10 submissions.
    - **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.

3. 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.

Additional Features

Plagiarism Checks:

- **Software Integration:** Use MOSS for plagiarism detection.
- **Automated Checks:** Automatically check each submission for code similarities.
- **Reporting:** Generate reports on potential plagiarism.

Cache Handling:

- **Caching Results:** Store results of previously evaluated submissions to avoid redundant computations.
- **Cache Invalidation:** Implement strategies to keep the cache up-to-date.
- **Performance Optimization:** Use caching to enhance system performance.

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.