**Online Judge**

**Problem Statement:**

In a coding challenge, participants compete among themselves by solving some coding questions, which are decided by the host of the contest, in a specified time frame. The participants are evaluated on the basis of the number of questions they answered correctly and also the time they took to solve the questions. An online judge is a platform that hosts such events.

A response for a question is said to be correct if the outputs of his/her code matches with the actual expected outputs for all the test cases.

A leaderboard is displayed at the end of the event, showing the relative performance of the participants.

**Overview:**

To design an online judge platform using MERN stack. The platform should be able to host multiple participants at a time. It should be able to able to take input code and evaluate it for the displayed and hidden test cases and display the result, whether it is accepted or not and save the progress of the participant.

**Features:**

Minimum expected features:

Account creation: Candidates should be able to create their accounts on the platform. They can do it using their email id or phone number or other means (using google account, Facebook account etc.)

User Registration: Candidates should be able to register for the contest.

Profile Management: Candidates should be able to view their profiles, which would contain their personal information, consist of history of contests they attended and the respective statistics, global ranking among all the users of the platform, information about the practice questions they solved on the platform etc.

Practice Problems: The platform should provide practice problems that do not contribute to the scoring or rankings.

Solution Submission: Participants should be able to submit their solutions to the problems during the competitions. They can upload their code or provide a text-based solution through the platform.

Solution Evaluation and Scoring: The platform should have a mechanism to evaluate the submitted solutions against the underlying test cases and generate scores.

Competition Leaderboard: Participants should be able to fetch the leaderboard of a specific competition. This leaderboard will display the rankings of participants based on their scores in that particular competition.

Additional features(Optional):

Premium Access: The platform should be able to provide addition features for its premium users

**Challenges:**

1. Load issues.

2. Should be resistant to attacks like sql attacks, scripting attacks etc.

3. Should not take more memory and ram.

4. Maintain authentication and authorization.

**Solutions:**

1. Rate limiting can be use to handle loads for a single system. Else we can deploy the application on a platform that provides scalability.
2. The application should be isolated from the server that runs it. This is to ensure safety from malicious attacks. Hosting on docker platform could help in this regard.
3. More memory requirement and time requirement, higher than usual, from a candidate could result in ability to host a greater number of concurrent users. This can be tackled by assigning limitations on both time and memory consumption by a user.
4. Differentiating between premium and non-premium users.

**High Level Design:**

1. Database Designing

● *Table 1: Problems*

Table structure:

Problem number: Number (Integer)

Problem Type: String (CharField)

statement: string (CharField)

name: string (CharField)

difficulty: string (CharField, optional)

● *Table 2: Submissions of the user*

Table structure:

Problem Number: reference to the problem document (Foreign Key)

User ID: Name of the user or any identifying detail. (CharField)(Foreign Key)

verdict: string (CharField)

submitted\_at: date and time (Auto DateTime Field)

● *Table 3: Test\_cases*

Table structure:

input: string (CharField)

output: string (CharField)

Problem Number: reference to the problem document (Foreign Key)

*● Table 4: Login/Signup*

UserId: string (CharField)

Password : string(CharField)

Email : string(CharField)

DOB : Date

FullName : string(CharField)

*● Table 5: Leaderboard*

Date : Date of the contest

UserId: string (CharField)

Score: Integer

Problems solved: Integer

Time taken: Integer

Position: integer

2. Web Server Designing:

● UI:

Screen 1: User Screen

Problem List

Login/Signup

Screen 2: Profile Screen

Problems solved

Password management

Performance in previous contests

Screen 3: Specific Problem

Language selection

File Selection

Coding Arena

Verdict / Submission Log

Input area

Run

Screen 4: Leaderboard

List of top performers.

**Screen Design descriptions (High level):**

● User Screen:

Frontend: Create a simple list UI in React that displays the names of each problem and links them to individual problem pages.

Backend: Define an API endpoint that handles a GET request to fetch all problems from the database and return them to the frontend.

● Profile Screen:

Frontend: Display the details of the user, previous performances, previous questions answered.

Backend: Define a template and API endpoint to handle a GET request to fetch the said details to the front end.

● Specific Problem page:

Frontend: Include a submit button below the code submission box in the "Show Individual Problem" template.

Backend: Define an API endpoint 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.
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.

● Leaderboard:

Frontend: Create a list to display the information of the top performers.

Backend: Define an API endpoint to handle a GET request for fetching the top performers data from the table.