**Online Judge (OJ)**

**Problem:**

A coding challenge is like a competition where people try to solve coding problems. They have a limited time, usually a few hours or days, to come up with solutions to these problems. People who sign up for the challenge compete by sending in their answers. These answers are then tested with hidden test cases to see if they work correctly. Depending on how well their answers do, participants get scores. An online judge is like a website that organises these challenges. It helps set up the contests, lets people submit their solutions, and checks if their solutions are right. Some examples of these websites are Codechef and Codeforces.

**Overview:**

Designing a Full stack online judge application using Django framework.

Takes different code from different users over the server and evaluates it automatically as accepted or not accepted.

**Features:**

Here are some key features which are expected in this application.

1. **User Registration:** Participants should be able to register for future coding challenges / events by providing their personal information such as name, email etc.
2. **User Login:** Participants should be able to login to their profile. And can attend current coding challenges or practice the coding problems.
3. **Dashboard:** when we sign in or sign up, the user will land on the dashboard page. Where the user can see the newly added first 10 problems.and he/she is able to solve that problem.
4. **Profile management:** Participants should be able to access their profile data and participants history. This allows them to track their progress and view their past competition performances.
5. **Practice Problem:** The platform should provide practice problems that do not contribute to the scoring or ranking. These problems allow participants to improve their skills and gain experience without the pressure of competition .
6. **Solution Evolution and Scoring:** This platform should have a mechanism to evaluate the submitted solutions against the underlying test cases and generate scores. This evaluation process should be automated to ensure fairness and accuracy in score.
7. **Leaderboard:** Participants should be able to fetch the leaderboard of a specific problem. This leaderboard will display participants submission log.

**Challenges:**

1. A scenario in which thousands of Users are Submitting the Solution at same time(Thundering Herd).
2. Someone uploads a code that has a malicious event
3. An unauthorised person gets access to manipulate the verdicts and output on the server.

**Solutions:**

To tackle this problem, we can use Rate limiting but it is generally considered as a bad practice.   
So, we can have a message queue in which we store the events to execute the code file at some time in the future. This will be an Asynchronous Process.

To Tackle this we use Docker that basically makes containers . We can easily assign each container a set amount of memory and test whether that code executes within that span of memory or not. It also provides a safeguard against any user trying to eat up memory with malicious code.

Isolating our core logic using Custom Isolation. We will implement this API using Docker.

**High Level Design :**

**Database Design:** To design a database for this application we will divide Tables into mainly 4 parts.

1. **User:**

UserId: VARCHAR(50)

Password: VARCHAR(50)

UserName: VARCHAR(255)

Email: VARCHAR(255)

DOB: DateField

1. **Problem**:

Id: INT

Number: VARCHAR(50)

Difficulty: VARCHAR(50)

Title: VARCHAR(255)

Statement: TEXT

Tags: VARCHAR(255)

ExampleInput: TEXT

ExampleOutput: TEXT

1. **SubmissionLogs**:

SubmissionId: INT

ProblemId: INT

UserId: INT

SubmittedAt: DATETIME

Language: VARCHAR(50)

Status: VARCHAR(50)

Code:TEXT

ErrorMessage: TEXT

1. **TestCases**:

Id: INT

ProblemId: INT

Input: TEXT

ExpectedOutput: TEXT

**Web Server Design:**

* **UI**
  + **NavBar:** 
    - **Home**
    - **Problems**
    - **Discuss (Optional)**
    - **About us**
    - **Profile**
    - **LogIn/LogOut**
* **Home Dashboard**
  + **Frontend:** This screen will contain newly added first 10 problems list
  + **Backend:** We will define a home route for this and this will take the first 10 newly added problems based on the created date from the database and render this home page.
* **User Profile:** 
  + **Frontend:** This screen will contain User detail, solved problem count, badge of problem solving skills. This screen also contains the user's last 5 submission logs.
  + **Backend:**  In backend we will extract all data for this user from User table and SubmissionLog table.
* **Problems List:**
  + **Frontend:** In this problem list, we will show all problems with links of each and individual problem links which will render to a single problem, there we can solve that problem.  
    In this list we will add filters based on difficulty level.
  + **Backend:** In the backend API we will extract all problems and render to that problem list page.
* **Show Problem** 
  + **Frontend:** Each problem will have one button that is “Solve”, to solve this problem we will click on this button and it will render to that problem page there we can read that problem and also solve that problem.
  + **Backend:** We will create an API to show that problem. This API will render that problem based on problem id.
* **Code Submission**
  + **Frontend:** After Solving that problem we will submit that code in the code section. There will be a “Submit” button.
  + **Backend:** Define an API endpoint to handle a POST request from the frontend. This endpoint should execute the following steps: 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. You can 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.
* **Leaderboard:** 
  + **Frontend:** we will display last 10 verdict (submission log) of each problem
  + **Backend:** We extract data from SubmissionLog for each problem. And render a list of the last 10 logs to that leaderboard page.

**Evaluation System :**

**Docker:**

Use special containers running on machines with high CPU to run the submitted code. Code sandboxing is necessary for the executions.

doesn’t consume too much of the resources should have the appropriate privileges set so that the code doesn’t peek into system config should have time limits se

**Other Features:**

* **Plagiarism Check** (Using software like MOSS)
* **Cache handling**