## Problem Statement :

A coding challenge is a competitive event in which a group of participants solve a set of coding questions within a specified timeframe, typically ranging from a few hours to a few days. Participants who have registered beforehand compete by submitting their solutions, which are evaluated against concealed test cases. Based on the test results, participants are assigned scores. An online judge is a platform that hosts these coding challenges, providing the infrastructure to manage and execute the competitions.

# Overview:

Designing a Full Stack Online Judge Using Mern Stack. Here users can try DSA problems and participate in contest. Evaluates it automatically as accepted or not accepted.

## Features in online Judge :

Here are some key features expected in the design:

**User Registration:** Participants should be able to register for future competitions by providing their personal details such as name, email, and password.

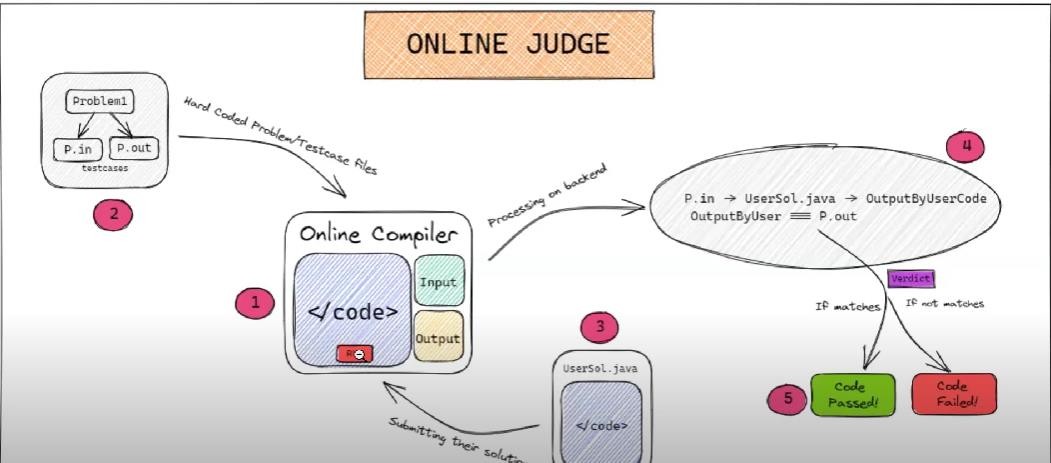
**Login :** After registering in online judge user have to login and tackle the dsa problems of Online judge.

**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.

**Profile Management:** Participants should have access to their profile, which includes personal details and their participation history and user see how much problem attempted. This allows them to track their progress and view their past competition performances.

**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.

**Practice Problems:** The platform should provide practice problems that do not contribute to the scoring or rankings. These problems allow participants to improve their skills and gain experience .



# High Level Design :

## Database Designing

* Collection 1: Login/Signup UserId: string (CharField) Password : string(CharField) Email : string(CharField)

DOB : Date

FullName : string(CharField)

* Collection 2: problems Document structure:

statement: string (CharField)

name: string (CharField) code: string (CharField)

difficulty: string (CharField, optional)

* Collection 3: solutions Document structure:

problem: reference to the problem document (Foreign Key) verdict: string (CharField)

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

* Collection 4: test\_cases

input: string (CharField) output: string (CharField)

problem: reference to the problem document (Foreign Key)

## Web Server Designing :

* **UI:**

Screen 1: Home Screen Practice Problems List Login/Signup

Screen 2: Specific Problem

Language selection

Coding Area

Verdict / Submission Log

Difficulty Level

Problem Point

Screen 3: LeaderBoard

List of top performers.

We will have a combination of routes and controllers.

* List Problems:

**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 in node.js that handles a GET request to fetch all problems from the database (MongoDB) and return them to the frontend.

* Show Individual Problem:

**Frontend**: Design a template in React to display the problem name, statement, difficulty level and a submission box for problem code in text format.

**Backend**: Define an API endpoint in node.js to handle a GET

request to fetch the problem details from the database and return them to the frontend.

* Code Submission:

**Frontend:** Include a run button to check sample test cases and submit button below the code submission box in the "Show Individual Problem" template.

**Backend:** Define an API endpoint in node.js 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:** Create a list UI in React to display the verdicts of the most problems submit by the specific user submissions.

**Backend:** Define an API endpoint in node.js to handle a GET request for fetching the solutions along with the verdicts for the most problem submissions from the database by specific user**.**

## Evaluation System :

**DOCKER :**

Use special containers running on machines with high CPU to run the submitted code. Code sand boxing is necessary so that 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 set

## Other Features :

**Plagiarism Checks(using softwares like MOSS) Cache Handling**