Online Judge

Problem Statement :

A coding challenge is a competitive event where registered participants solve coding questions within a set time frame. They submit solutions to an online judge platform (e.g., Codechef, Codeforces), which evaluates their solutions against hidden test cases, assigning scores based on the results.

Overview:

Creating a Full Stack Online Judge using the MERN stack involves building a platform that allows users to submit code via the server. The system then automatically evaluates the code, determining whether it meets the acceptance criteria or not.

Features :

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.

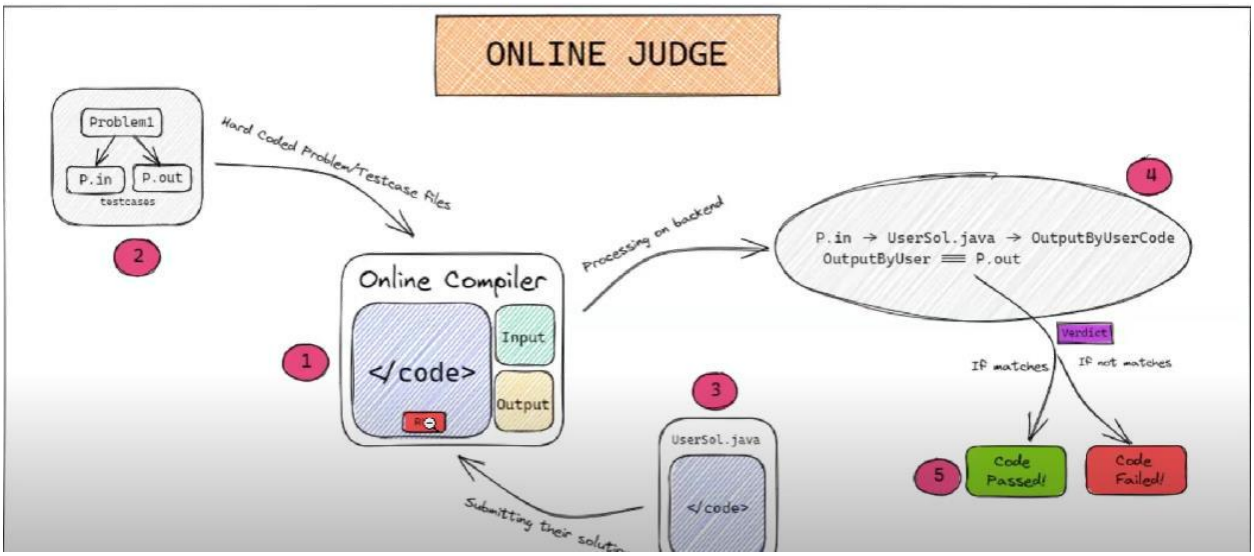
Solution Submission: Participants should be able to submit their solutions to the problems during the competitions. They can upload their code or provide a text-based solution through the platform.

Profile Management: Participants should have access to their profile, which includes personal details and their participation history. 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 hone their skills and gain experience without the pressure of competition.

Solution Evaluation and Scoring: The 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 scoring.



High Level Design :

1. Database Designing

Entities involved are:

1) user
2) problem
3) submission
4) discussion
5) testcase
6) favorite problem
7) leaderboard

Relationships among entities:

User and Favorite Problem:
One-to-Many: A user can have many favorite problems.
Relationship: User (one) - FavoriteProblem (many)

User and Submission:
One-to-Many: A user can have multiple submissions.

Relationship: User (one) - Submission (many)

Problem and Submission:
One-to-Many: A problem can have multiple submissions.
Relationship: Problem (one) - Submission (many)

Problem and Test Case:
One-to-Many: A problem can have multiple test cases.
Relationship: Problem (one) - TestCase (many)

Problem and Discussion:
One-to-Many: A problem can have multiple discussions.
Relationship: Problem (one) - Discussion (many)

User and Leaderboard:
One-to-One: Each user corresponds to a single leaderboard entry.
Relationship: User (one) - Leaderboard (one)

Schema Design:

1.) User Schema:
This schema stores information about the users who use the online coding judge.
{
“Username” => String
“Email” => String
“Password” => String
“Profile Picture” => String(base64 encoded)
}

2.) Problem Schema:
This schema stores information about coding problems available on the platform.

{
"Title" => String
"Description" => String
"Difficulty" => String
}

3.) Submission Schema:
This schema stores information about user submissions for a particular problem.

{
"User" => Object
"Problem" => Object
"Code" => String
"Language" => String
"Status" => Boolean
"Submission Time" => String
}

4.) Discussion Schema:
This schema stores discussions related to a specific problem, where users can ask questions and provide answers or comments.

{
"Problem" => Object
"User" => Object
"Text" => String
}

5.) Test Case Schema:
This schema stores information about test cases associated with a particular problem.

{

“Problem” => Object
“Input” => String
“ExpectedOutput” => String
}

6.) Favorite Problem Schema:
This schema allows users to mark problems as their favorites.
{
“User” => Object
“Problem” => Object
}

7.) Leaderboard Schema:
This schema keeps track of user rankings and scores.
{
“User” => Object
“Score” => Number
}

2. Web Server Designing :

2.A) API EndPoints

User API Endpoints:

Register User: POST /api/users/register
Login User: POST /api/users/login
Logout User: POST /api/users/logout
Get User Information: GET /api/users/:userId

Update User Details: PUT /api/users/:userId
Delete User: DELETE /api/users/:userId

Problem API Endpoints:

Create Problem: POST /api/problems
Get Problem: GET /api/problems/:problemId
Get All Problems: GET /api/problems
Update Problem: PUT /api/problems/:problemId
Delete Problem: DELETE /api/problems/:problemId

Submission API Endpoints:

Submit Solution:POST /api/submissions
Get Submission: GET /api/submissions/:submissionId
Get Submissions for Problem:
GET /api/submissions/problem/:problemId
Get Submissions for User:
GET /api/submissions/user/:userId

Favorite Problem API Endpoints:

Add Favorite Problem: POST /api/favorites
Remove Favorite Problem:
DELETE /api/favorites/:favoriteId
Get Favorite Problems for User:
GET /api/favorites/user/:userId

Test Case API Endpoints:

Create Test Case: POST /api/testcases
Get Test Cases for Problem:
GET /api/testcases/problem/:problemId

Discussion API Endpoints:

Create Discussion: POST /api/discussions
Get Discussions for Problem:
GET /api/discussions/problem/:problemId

Leaderboard API Endpoints:

Update User Score: PUT /api/leaderboard/user/:userId
Get Leaderboard: GET /api/leaderboard

2.B) UI in ReactJS

1) HomePage
2) Login Page
3) Registration Page
4) Dashboard
5) Browse Problems Page
6) Problem Details Page
7) Profile Page
8) LeaderBoard Page
9) Favorite Problems Page
10) Discussion Page

3. 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 :

1.)Plagiarism Checks(using softwares like MOSS)
2.)Cache Handling
3.)Message Queue to handle large submissions.