ONLINE JUDGE

PROBLEM STATEMENT:

*To make an Online Judge Web Application.*

Online judge application is a platform where users solve a particular coding problem using their desired coding language. Based on the solution provided by a user online judge will evaluate the correctness and efficiency of submitted solution against some predefined test-cases. Online Judge Application also hosts coding contests/challenges and provides Rating/Leaderboard for contest participants.

Overview:

Designing and Developing a Full Stack Online Judge Using Mern Stack.

Users will submit the code over the server for a particular problem and Server evaluates it over certain testcases based on which a verdict is sent back to user.

**Features :**

Here are some key features expected in the design:

**User Registration**: User should be able to register on Online Judge Application

In-order to keep a track of solved problems and to participate in challenges.

**Solution Submission**: Users should be able to submit their

code to the problems during the competitions. They can upload

their code or provide a text-based solution through the platform.

**Profile Management**: Users 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**: Users 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 enhance their problem-solving skills.

**Solution Evaluation and Scoring**: The platform should have a

mechanism to evaluate the submitted solutions against the underlying

test cases. This evaluation process should be automated to ensure fairness and

accuracy in scoring.

**Challenges** :

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

**Solutions** :

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}

I will develop this web application using MERN stack.

* MongoDB - For Database.
* Express.Js - For creating HTTP server.
* React - For creating Dynamic user interface(Frontend).
* Node.Js - For creating Backend of the application.

**1.DATABASE DESIGN**

* USER COLLECTION
* UserId
* Username
* Password
* Email
* Date Of Birth
* Institute
* ADMIN COLLECTION
* UserId
* Username
* Password
* Problem-Id
* PROBLEM COLLECTION
* Problem Id
* Problem statement
* Difficulty level
* Code
* SOLUTION COLLECTION
* Problem Id
* Verdict
* SubmittedAt
* Solution Id
* TESTCASE COLLECTION
  + Testcase-Id
  + Input
  + Output
  + Problem-Id

**2.WEB SERVER DESIGN**

* **USER INTERFACE**
* Screen 1->Home Screen
* Problem List
* Login/Signup
* Screen 2->Specific Problem
* Coding playground
* Language selection
* Verdict/Submission log
* Screen 3-> Leaderboard(Optional)
* List of top performers.
* 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 Express.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, 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.

● 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:

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

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:**

**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