**Online Judge System**

**Introduction**

The system will allow users to register, log in, view coding problems, submit solutions, and receive feedback. The system will have a frontend interface for user interaction and a backend system to handle requests, manage data, and evaluate submissions.The system will be built using the MERN stack: MongoDB, Express.js, React, and Node.js.

**Architecture Overview**

The system will be divided into three main components:

1. Frontend: A React application for user interaction.
2. Backend: An express.js , Nodejs server for handling API requests.
3. Database: MongoDB (Atlas) for data storage.

**Features**

Here are some key features expected in the design:  
  
1. **User Registration and Authentication**

- Users can register and log in using their email and password.  
 - JWT tokens will be used for session management.

2. **Problem Management**  
 - Admins can add, update, and delete programming problems.  
 - Problems will have details like title, description, input/output format,  
 constraints, and example cases.

- Have leaderboard section where it shows last 10 submissions along   
 with verdict   
  
3. **Submission and Evaluation** - Users will have to login to submit their code for a problem   
 - The system will compile and run the code against predefined test   
 cases.  
 - Feedback will be provided based on the output (e.g. Accepted, Not   
 accepted, time limit exceeded).  
  
4. **User Profiles** - Users can view their submission history and problem-solving statistics.

**Non-Functional Requirements**

1. **Scalability**: The system should handle a growing number of users and submissions.
2. **Performance**: The evaluation of submissions should be efficient.
3. **Network Isolation**: User data and code submissions should be securely stored and transmitted.

**High Level Design :**

1. **Database Designing**

**Collection 1 : Login/SignUp**

* **name**: String
* **email**: String
* **password**: String (hashed)
* **role**: String (user/admin)

**Collection 2 : Problems**

* **statement**: String
* **question**: String
* **Examples case**: Array
* **Constraint**: string

**Collection 3 : Submission**

* **problem**: reference to the problem document (Foreign Key)
* **verdict**: String
* **submitted\_at:** date and time (Auto DateTime Field)
* **code:** string
* **language:** string

**Collection 4 : test\_cases**

* **problem**: reference to the problem document (Foreign Key)
* **input**: String
* **output:** string

**2. Web Server Designing**

UI Screen 1:

* Login/Signup

UI Screen 2:

* Problems List
* Specific Problem
* Language selection
* File Selection
* Coding Arena
* Submission button
* Leaderboard section

UI Screen 3:(User Profile)

* User submitted problems

**3. API Endpoints**

* Login/Signup

**Frontend:** Create a simple list UI in React that allows for new user to register and login for existing users

**Backend:** Define an API endpoint in Express.js that handles a

POST request to store/fetch the details from/in the database and return them to frontend

* Listing all the 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:  
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 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**.**

**4. Evaluation System**

A separate service (or worker) will be responsible for compiling and running user submissions against test cases.The service will return the result (pass/fail) and any relevant messages.Should have time limit set so that executions doesn’t consume too much of resources