ONLINE JUDGE

Problem Statement

Create an online judge website where users can submit their code and get a verdict in return, based upon the code provided for a problem. This online judge will consist of multiple coding problems and an online compiler where users will write their code and can run the code with their custom test cases or provided sample test cases. A user must create an account in order to be able to use the website so that the progress of each user can be tracked and provided to them.

Overview

The platform will support multiple programming languages and provide a comprehensive, interactive experience for users to improve their coding skills.The website will be developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) and will be deployed on the cloud for accessibility from any system.

Key Features

- **Authentication:** Participants must register to the website before attempting a problem.
- **Problem Submission and Evaluation:** When a solution to a problem is submitted, a verdict is returned to the user based on the solution.
- **Code Collaboration:** Allow users to work on problems together in real time, with shared code editing and chat functionality.
- **Leaderboard:** Users are arranged on the basis of the number of problems they have solved.
- **User Profile Tracking:** Users can keep the track of their progress.

Frontend Design

- **Home Page**
    - **Purpose**: Display a list of problems, an overall leaderboard, and options for user authentication.
    - **Components**:
        - **Header**: Contains options to log in/sign up.
        - **Problem List**: Displays the available problems.
        - **Leaderboard**: Shows the number of problems solved by users and the number of attempts taken.
- **Login/Signup Page**
    - **Purpose**: Allow users to log in to their accounts using their username and password.

    - **Components**:
        - **Header**: Contains the logo and navigation links to the home page
        - **Login form**: Username Field, Password Field, and Submit Button.
        - **Sign-Up Form**: Name Field, Username Field, Password Field, and Create Button.
- **Problem Page**
    - **Purpose**: Provide the problem description and allow users to write, test, and submit their code.
    - **Components**:
        - **Problem Description**: Includes I/O explanation and example test cases.
        - **Code Editor**: Window for writing code.
        - **Input Window**: Separate window for entering test cases.
        - **Output Window**: Displays the results of the code execution.
        - **Buttons**: 'Run' button to execute the code and 'Submit' button to check against hidden test cases.
        - **Code Collaboration Button**: Redirects to a page for real-time code collaboration..
- **User Profile Page**
    - **Purpose**: Display user's basic details and track their progress.
    - **Components**:
        - **Profile Information**: Displays name, username, and other basic details.
        - **Progress Tracking**: Shows progress with graphical representations.
- **Code Collaboration Page**
    - **Purpose**: Facilitate real-time code collaboration between users.
    - **Components**:
        - **Shared Code Editor**: Allows multiple users to edit the same code simultaneously.

Backend Design

The backend will be developed using Node.js and Express.js, providing a robust and scalable server-side application. Key functionalities include user authentication, problem management, code execution, and data storage.

REST APIs

1. **/login**
    - **Request Body:**

      { username: string, password: string }

    - **Response Body:**

      { success: true, token: string, user: { id: string, username: string, name: string } }

- **Functionality:** Authenticates the user, generates a JWT token, and returns user details.

2. **/signup**
   - **Request Body:**

     { username: string, password: string, name: string }

   - **Response Body:**

     { success: true, user: { id: string, username: string, name: string } }

   - **Functionality:** Registers a new user and returns the user details.

3. **/problems**
   - **Request Body:** None
   - **Response Body:**

     { problems: [{ id: string, title: string, description: string, difficulty: string }] }

   - **Functionality:** Fetches and returns a list of available problems.

4. **/problem/**
   - **Request Body:** None
   - **Response Body:**

     { id: string, title: string, description: string, input: string, output: string, exampleTestCase: { input: string, output: string } }

   - **Functionality:** Fetches and returns the details of a specific problem.

5. **/submit**
   - **Request Body:**

     { problemId: string, code: string, language: string }

   - **Response Body:**

     { success: true, verdict: string, details: { passed: number, failed: number, total: number } }

   - **Functionality:** Submits the user's code for evaluation and returns the verdict.

6. **/profile**
    - **Request Body:** None
    - **Response Body:**

      { user: { id: string, username: string, name: string, solvedProblems: number, attemptedProblems: number } }

    - **Functionality:** Fetches and returns the user's profile details and progress.

Database Design

1. **Users:**

   _id: ObjectId

   username: string

   password: string

   name: string

   solvedProblems: array

   attemptedProblems: array

2. **Problems:**

   _id: ObjectId

   title: string

   description: string

   input: string

   output: string

   exampleTestCase: { input: string, output: string }

   testCases: array

3. **Submissions:**

   _id: ObjectId

   userId: ObjectId

   problemId: ObjectId

code: string

language: string

verdict: string

timestamp: date

Full Tech Stack Flow

1. **Frontend**:
    - **React.js**: Create components such as problem listing (**ProblemList**) and problem details (**ProblemDetails**).
    - **React Router**: Navigate between different pages (home, problem details, user profile, etc.).
2. **Backend**:
    - **Express.js**: Define routes and controllers for fetching problems and handling submissions.
    - **Mongoose**: Interact with MongoDB to fetch and store problem and submission data.
3. **Database**:
    - **MongoDB**: Store collections for **Users**, **Problems**, and **Submissions**.
4. **Code Execution**:
    - **Docker**: Docker will be used to containerize the application, ensuring consistent environments for development, testing, and production.