**Sahil Kabra Online Judge HLD Document**

**About the Project:**  
The Online Judge platform is designed to offer coding enthusiasts and developers a space to practice coding problems, participate in contests, and hone their programming skills. It aims to provide an easy-to-use interface and a strong backend system for a smooth problem-solving experience. It also gives a chance to test yourself against the best through weekly contests and leaderboard rankings for the same as well as overall platform rating. It is designed to be a secure platform wherein user data and privacy are of paramount importance. It will contain MERN stack for front end, node.js & express.js for backend, as well as MongoDB as the NoSQL database. The project is containerised using dockers to ensure consistent environments. AWS provides the structure for deployment to cater to scalability and reliability.

**Front End/UI:**

The Online Judge platform's frontend is built to be easy to use and interactive. It's made with React.js, which helps in creating a smooth and dynamic user experience. The design is modular, meaning different parts of the app are managed separately but work together.

Detailed Screen Breakdown

1. Authentication Screen

Purpose: Allow users to log in or register to access the platform.

Components:

* Login Form: For existing users to log in.
* Register Form: For new users to create an account.

Functionality: Users can switch between login and registration forms. Form validation for input fields. Submission of login or registration details to the backend.

Security Considerations:   
Security measures protect user data and ensure safe interactions with the backend.

* Authentication: Uses JWT tokens for secure login.
* Authorization: Role-based access control to restrict certain areas.

2. Dashboard Screen

Purpose: Provide an overview of user-specific information.

Components:

* User Progress: Displaying solved problems and progress.
* Suggested Problems: Recommendations based on user history.
* Recent Activities: List of recent submissions and activities.

Functionality: Fetch and display user-specific data from the backend.

3. Problem List Screen

Purpose: Display a list of available coding problems for users to choose from.

Components:

* Problem Filters: By difficulty, tags, search bar.
* Problem Cards: Each problem displayed with title, difficulty, and brief description.

Functionality: Fetch and display problem data. Enable filtering and searching of problems.

4. Problem Details Screen

Purpose: Show details of a specific coding problem.

Components:

* Problem Statement: Title, description, input/output format, constraints, examples.
* Code Editor: Integrated editor for writing code.
* Language Selector: Dropdown for selecting the programming language.

Functionality: Fetch and display detailed problem information. Allow users to write and edit code. Enable language selection.

5. Code Submission Screen

Purpose: Enable users to submit their code solutions and receive feedback.

Components: Code Editor: Integrated editor for writing and editing code. Language Selector: Dropdown for selecting the programming language. Submit Button: Button to submit the code. Submission Results: Display results of the code submission. Functionality: Users can write or paste their code and select the language. Submit code to the backend and display real-time feedback.

6. Profile Screen

Purpose: Display user profile information and submission history.

Components: User Info: Username, email, other profile details. Solved Problems: List of solved problems. Submission History: Detailed history of past submissions. Functionality: Fetch and display user profile data from the backend. Enable profile updates.

7. Leaderboard Screen

Purpose: Display user rankings based on contest performance. Components:

* Leaderboard Table: Showing user rankings, including username, score, rank.
* Filters: Options to filter by contest, date range, or other criteria.

Functionality: Fetch and display leaderboard data. Fetches leaderboard data from the backend using a GET request. Displays rankings in a tabular format. Allows users to filter and sort the leaderboard based on different criteria.

**Back End:**

The backend of the Online Judge platform is built using Node.js and Express.js. Node.js is a JavaScript runtime environment that allows for the execution of JavaScript code on the server side. Express.js is a lightweight and flexible web application framework for Node.js, which provides robust features for building web and mobile applications. Together, Node.js and Express.js offer an efficient and scalable environment for handling the server-side logic, routing, and API integrations of the platform.

REST APIs:

In a RESTful API, the request and response body are the data payloads sent by the client to the server (request) and by the server to the client (response). These bodies are in JSON format for easier parsing and understanding.

1. Authentication APIs

/register

Purpose: Register a new user.

Request Body:  
{

"username": "string",

"email": "string",

"password": "string"

}

Response Body:  
{

"message": "User registered successfully",

"userId": "string"

}

/login

Purpose: Authenticate a user and generate a JWT.

Request Body:  
{

"email": "string",

"password": "string"

}

Response Body:  
{

"message": "Login successful",

"token": "string"

}

/logout

Purpose: Log out a user (client-side can handle token removal).

Request Body: None

Response Body:  
{

"message": "Logout successful"

}

2. User APIs

/users/

Purpose: Get user profile information.

Request Body: None

Response Body:  
{

"userId": "string",

"username": "string",

"email": "string",

"solvedProblems": ["array of problem IDs"],

"submissionHistory": ["array of submission IDs"]

}

/users/update

* Purpose: Update user profile information.

Request Body:  
{

"username": "string",

"email": "string"

}

Response Body:  
{

"message": "User profile updated successfully"

}

3. Problem APIs

/problems

Purpose: Get a list of coding problems.

Request Body: None

Response Body:  
{

"problems": [

{

"id": "string",

"title": "string",

"difficulty": "string",

"tags": ["array of strings"],

"description": "string"

},

...

]

}

/problems/

Purpose: Get details of a specific problem.

Request Body: None

Response Body:  
{

"id": "string",

"title": "string",

"difficulty": "string",

"tags": ["array of strings"],

"description": "string",

"inputFormat": "string",

"outputFormat": "string",

"constraints": "string",

"examples": [

{

"input": "string",

"output": "string"

},

...

]

}

4. Submission APIs

/submissions

Purpose: Submit code for a problem.

Request Body:  
{

"userId": "string",

"problemId": "string",

"code": "string",

"language": "string"

}

Response Body:  
  
{

"message": "Submission received",

"submissionId": "string",

"status": "string"

}

/submissions/

Purpose: Get the result of a submission.

Request Body: None

Response Body:  
{

"submissionId": "string",

"problemId": "string",

"userId": "string",

"code": "string",

"language": "string",

"status": "string",

"result": {

"passedTests": "number",

"failedTests": "number",

"errorMessages": ["array of strings"]

}

}

#### 5. Leaderboard APIs

/leaderboard

Purpose: Get the leaderboard rankings for contests.

Request Body: None

Response Body:  
{

"leaderboard": [

{

"rank": "number",

"userId": "string",

"username": "string",

"score": "number",

"contestId": "string"

},

...

]

}

5. Leaderboard APIs

/leaderboard/

Purpose: Get the leaderboard rankings for a specific contest.

Request Body: None

Response Body:  
  
{

"contestId": "string",

"leaderboard": [

{

"rank": "number",

"userId": "string",

"username": "string",

"score": "number"

},

REST APIs cover user authentication, user management, problem management, code submission, and leaderboard rankings, each with clearly defined request and response bodies to facilitate smooth interaction between the frontend and backend.

**Database:**

MongoDB Schema Design MongoDB, being a NoSQL database, uses collections instead of tables. Each collection contains documents, which are analogous to rows in relational databases but with flexible schemas.

1. Users Collection Purpose: Store user information such as username, email, and other profile details. Fields: userId: Unique identifier for each user. username: User's chosen username. email: User's email address (used for login and contact).

#### 2. Problems Collection

Purpose: Store details of coding problems available on the platform.

Fields:

* pid: problem id
* pt: problem title
* ps: problem statement
* pd: problem difficulty (e.g., Easy, Medium, Hard).

#### 3. Test Cases Collection

Purpose: Store test cases associated with each problem for validating submissions.

Fields:

* tid: testCaseId
* Pid: problem id for which the test case is defined

#### 4. Submissions Collection

Purpose: Store user submissions of code solutions to problems.

Fields:

* userId: Reference to the user who made the submission.
* problemId: Reference to the problem for which the submission was made.
* testCaseId: Reference to the test case against which the submission was tested.
* code: Submitted code solution.

**Docker/Security Flaws/Solutions:**

Docker is used in the Online Judge platform to containerize the application, ensuring consistency and portability across different environments.

Containerization: Each component of the platform (frontend, backend, database) is encapsulated within Docker containers.

Isolation: Docker containers ensure isolation between components, preventing conflicts and minimizing dependencies on the host system.

Scalability: Docker allows easy scaling by spinning up multiple containers of each service, facilitating load balancing and performance optimization.

#### Security Measures

1. Protection from DDoS Attacks:

* Implement rate limiting and request validation to mitigate DDoS attacks.
* Utilize AWS Shield or third-party services for DDoS protection and mitigation.

2. User Permissions:

* Define roles (admin, user, moderator) with specific access levels to different features and data.

3. User Data Protection:

* Encrypt sensitive user data using HTTPS and encryption algorithms.
* Store passwords securely using strong hashing algorithms (e.g., bcrypt).

4. Protection against Malicious Code and Scripts:

* Implement input validation and sanitization to prevent injection attacks (SQL injection, XSS).

5. File System Security:

* Secure the file system containing test cases and problem statements using file permissions and access controls.
* Regularly scan for vulnerabilities and apply security patches to the underlying operating system and Docker containers.

6. Network Isolation:

* Configure Docker networks to isolate frontend, backend, and database containers.
* Utilize AWS Virtual Private Cloud (VPC) for additional network isolation and security.

### **Deployment on AWS (EC2, VPC, ELB, S3, ECR, etc)**

Deploying the Online Judge platform on AWS involves leveraging various services for scalability, reliability, and performance.

1. Amazon EC2 (Elastic Compute Cloud):

* Provision EC2 instances to host Docker containers for frontend, backend, and database services.
* Choose instance types based on compute requirements and expected traffic volume.

2. Docker Container Deployment:

* Build Docker images for frontend (React.js), backend (Node.js with Express), and database (MongoDB).
* Push Docker images to Docker Hub or use Amazon ECR (Elastic Container Registry) for private repository.

3. Load Balancing:

* Configure Elastic Load Balancing (ELB) to distribute incoming traffic across multiple EC2 instances.

4. Auto Scaling:

* Set up Auto Scaling groups to automatically adjust the number of EC2 instances based on traffic patterns and application load.