HASH CODE - ONLINE JUDGE APPLICATION 2024

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. It allows users to check their code against custom inputs before finally submitting it for evaluation. After successfully solving the problem their submission gets stored for future reference.

Overview:

*HASH CODE is an online judge application designed to provide a platform for users to solve programming problems and receive immediate feedback on their code submissions. The system is composed of several key components :*

1. User Interface (UI): The frontend component responsible for presenting the user interface, including the code editor, problem statements, and submission forms.
2. Backend Server: The backend component responsible for handling user authentication, code submission evaluation, and database operations.
3. Database: Stores user data, problem statements, test cases, and submission results.
4. Admin Panel: A separate interface for administrators to manage problem statements, users, and submissions.

**Architecture :**

*The architecture of HASH CODE follows a client-server model:*

* Client: The client-side application is built using HTML, CSS, and JavaScript, utilizing frameworks like React.js for dynamic UI rendering.
* Server: The server-side application is built using Node.js with Express.js framework. It handles HTTP requests from the client, authenticates users, evaluates code submissions, interacts with the database, and serves API endpoints.
* Database: The application uses a NoSql database MongoDB which provides with scalability, flexibility and quick-easy implementation.

**Components :**

1. User Interface (UI):

* Register/Login: Forms to register and login users .
* Problem List: A list of questions along with their difficulty levels.
* Code Editor: A code editor powered by NPM allowing users to write, edit, and test their code.
* Problem Statements: Display problem statements fetched from the backend, along with input/output specifications.
* Leaderboard: Users can see their rank among all the users registered in this web application
* Run/Submission Form: Allows users to either run their code for custom inputs or submit for evaluation.
* Admin: Admin login form, crud operation features.

1. Backend Server:

* Authentication: Handles user authentication.
* Code Evaluation Engine: Executes user-submitted code against test cases and returns the results.
* Database Access Layer: Manages CRUD operations for users, problem statements, submissions, and other entities.
* API Endpoints: Exposes RESTful APIs for client-server communication.

1. Database:

* User Model: Stores user credentials and timestamps.
* Question Model: Stores title, problem statements, input/output specifications, and other metadata.
* Submission Model: Logs user last successful submission, including Question title and timestamps.
* Admin Model: Stores admin credentials.

**Features:**

*Here are some key features expected in the design:*

* **User Registration**: User should be able to register on HASH CODE In-order to keep a track of solved problems.
* **User Login:** User should be able to login themselves once registered.
* **Solution check:** User should be able to check their code for custom inputs and receive an output to know the working of their code.
* **Solution Submission**: Users should be able to submit their code to the problem statements.
* **View Submission:** User should be able to view their last successful submission for future reference.
* **Language Selection:** User should be able to choose preferred coding language from the options provided.
* **Admin Login:** Admin should be able to login through provided credentials.
* **Admin Add Problem:** Admin should be able to add new problem statement with suitable testcases and outputs.
* **Admin Edit/Delete Problem:** Admin should be able to edit the problem statement also should be able to delete them.
* **Leaderboard**: Users can see their rank among all the users registered in this web application

**Challenges:**

* An unauthorized person gets access to manipulate the verdicts and output on the server.
* User continuously send request to api.
* User wants to create more than one id with similar username or email.

**Solutions:**

* Routes will be secured with Admin credentials which only renders while Admin session is ongoing
* Loader feature will be implemented which will freeze the screen and shows a loader until response is received.
* UserId and Username will be defined unique in the database and further checks will be implemented through controllers.

{HIGH LEVEL DESIGN}

*HASH CODE will be developed 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
* Email
* Password
* ADMIN COLLECTION
  + AdminId
  + Name
  + Password
* Question COLLECTION
* Ouestion\_Id
* Title
* Problem statement
* Difficulty level
* Input
* Output
* SUBMISSION COLLECTION
* User\_Id
* Submission
* Question\_id
* Submission\_Id
* CONTACT COLLECTION
  + Contact\_Id
  + Personal\_email
  + Query

**2.WEB SERVER DESIGN**

* **USER INTERFACE**
* Screen 1->Home Screen
* Navbar
* Introduction
* Screen 2->Register/Login
* Register form
* Login form
* Screen 3->Problem List
* List of questions with difficulty level
* Screen 4-> Particular Problem
* Problem Statement
* Submission
* Code Editor
* Custom Input
* Output
* **LIST PROBLEMS**
* **Frontend:** Create a simple list UI in React that displays the names of each problem with their difficulty level 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 theproblem 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.
* **Future Enhancements:**
* Support for additional programming languages and compilers.
* Integration with version control systems (e.g., Git) for code submissions and version tracking.
* Real-time collaboration features for pair programming and code reviews.