Code Quest

Introduction:

Code Quest is an online coding judge platform for coding challenges, offering users a space to solve problems, review solutions, and monitor their progress on leaderboards. This document details the complete development strategy, encompassing frontend and backend development, database design, and Docker configuration.

Frontend:

The frontend will be built using modern web technologies like HTML, CSS, and JavaScript, along with a frontend framework React.js

Pages:

Signup/Register Page: A user interface for new users to create an account.

Features:

• Username, Email, Contact number and Password fields.

- Validation checking for all the input fields

• Signup button.

• Link to the Login page.

UI:

• Input fields - username, email, contact number and password.

• Button for account creation.

• A link to navigate back to the login page.

Signin/Login Page: A user interface for users to log in to their accounts.

Features:

• Email and Password fields.

- Checking for correct password

• Login button.

• Link to the Signup page.

UI :

• Input fields - email and password.

• Buttons for login and navigation to the signup page.

Problems Page: Displays set of all available coding problems.

Features:

• List of problems with brief descriptions.

• Filters and search functionality (difficulty, tags).

UI

• A searchable and filterable list of problems.

• Click button to solve a particular button

Solution Page: Displays the details of a specific problem and be able solve

Features:

• Problem statement.

• Input/output format.

• Constraints.

• Code editor for submitting solutions.

• Coding language selection (C++/Java/Python)

• Submit button.

• Sample test cases.

UI Elements:

• Sections for problem details, sample test cases, and a code editor.

• Selection of coding language.

• Button for submitting the solution.

Leaderboard Page: Shows the ranking of users based on their performance.

Features:

• User ranking.

• Score

UI:

• A table displaying user rankings and points.

• Filters for different time frames.

User Profile Page :Displays the user's profile information

Features:

• Username, Email, Contact number.

• Profile picture(can upload here).

• List of solved problems.

• Performance statistics.

UI :

• Sections for user details, solved problems, and performance stats.

• Option to edit profile information.

Backend

The backend will be built using Node.js with Express.js framework, providing a RESTful API to interact with the frontend and the database.

The output of all the routes will be sent in JSON Format

Routes

1. Auth Routes

• **/api/auth/login**: POST - Login a user.

Authenticates the user and returns a session token.

• **/api/auth/signup**: POST - Register a new user.

Registers a new user and returns a success message.

• **/api/auth/logout**: POST - Logout the user.

Invalidates the user's session token.

2. User Routes

• **/api/users/:userId**: GET - Fetch user profile.

• **/api/users/:userId: PUT** - Update user profile.

3. Problem Routes

• **/api/problems**: GET - Fetch all problems.

• **/api/problems/:problemId**: GET - Fetch a single problem.

4. Submission Routes

• **/api/submissions**: POST - Submit a solution.

• **/api/submissions/:submissionId**: GET - Fetch submission status and results.

5. Leaderboard Routes

• **/api/leaderboard:** GET - Fetch leaderboard data.

Controllers

1. Auth Controller

Handles user login, sign up by validating credentials, creating a new record and returning a session token.

Handles user logout by invalidating the session token.

2. User Controller

Retrieves and updating of user profile information based on the provided user ID.

3. Problem Controller

Retrieves and returns a list of all coding problems and specific problems

4. Submission Controller

Handles code submission by storing the solution and initiating evaluation and returns the verdict of code

5. Leaderboard Controller

Retrieves and returns leaderboard data based on user performance.

Database

The database will be designed using MongoDB.

Schema Design

1. Users Collection

Schema:{

"_id": "ObjectId",

"username": "String",

"email": "String",

"password": "String",

"contact number": "String",

"profile_picture": "String",

"created_at": "Date"

}

2. Problems Collection

Schema:{

"_id": "ObjectId",

"title": "String",

"description": "String",

"input_format": "String",

"output_format": "String",

"constraints": "String",

"sample_tests": [

{

"input": "String",

"output": "String"

} : (Array of sample test cases, each containing input and output.)

],

"difficulty": "String"

}

3. Submissions Collection

Schema:

{

"_id": "ObjectId", (Primary key)

"user_id": "ObjectId", (Foreign key)

"problem_id": "ObjectId",

"code": "String",

"status": "String", (Submission status like pending , accepted or wrong answer

"created_at": "Date"

}

4. Leaderboard Collection

Schema:

{

"_id": "ObjectId", (Primary Key)

"user_id": "ObjectId",

"points": "Number",

"rank": "Number",

"created_at": "Date"

}

Docker Setup

Docker will be used to containerize the application to ensure consistency across different environments. Here are the steps involved:

1. Dockerfile

o environment for the application using an official Node.js image.

o Install dependencies and copy the application source code.

o Expose the application port and specify the startup command.

2. docker-compose.yml file

o services for the application and MongoDB.

o environment variables and port mappings.

o volumes to persist MongoDB data.

3. Build and Run Containers

o build the application image.

o Docker Compose to start both the application and MongoDB services.