CODE IT

Technology Stack: MERN (MongoDB, Express.js, React.js, Node.js)

Description:

Code It is a comprehensive coding challenge platform designed for developers to solve algorithmic problems and compete with others. Users can view a variety of coding problems and submit their solutions in C++, Java, or Python. The platform also provides functionalities to track submitted problems and view a leaderboard to see how their performance compares with other users.

Features:

- Sign up and login functionality using JWT for secure authentication.
- Profile management for users to update their personal information and preferences.
- Browse and search for coding problems categorized by difficulty, tags, and topics.
- Detailed problem descriptions with input/output examples and constraints.
- Automated evaluation of submitted code against predefined test cases.
- Real-time feedback on code correctness, performance, and edge cases.
- Detailed reports for each submission including runtime, memory usage, and test case results.

- A dynamic leaderboard displaying top performers based on the number of problems solved and efficiency.

Schemas:

- **User :** The User schema stores information about each user registered on the platform.

```
{
  _id: ObjectId,
  username: String,
  email: String,
  password: String,
  contact_number: String,
  profile_picture_url: String,
  created_at: Date
}
```

- **Problems :** The Problem schema defines the structure for storing coding problems available on the platform.

```
{
 _id: ObjectId,
title: String,
 description: String,
 input_format: String,
output_format: String,
 example_cases: [
        {
              input: String,
              output: String
       }
 ],
 difficulty: String,
 tags: [String],
solution;String,
created_at: Date
```

}

- **Submissions** : The Submission schema stores information about each solution submission made by users. It includes fields for the user who made the submission, the problem to which the solution was submitted, the programming language used, the submitted code, the status of the submission, execution details, and the date of submission.

```
{
  _id: ObjectId,
  user_id: ObjectId,
  problem_id: ObjectId,
  language: String,
  code: String,
  status: String,
  execution_time: Number,
  memory_usage: Number,
  submitted_at: Date
}
```

API Interface:

→ User Endpoints

1.Register User :

POST /api/users/register

Description: Register a new user with a username, email, and password.

2.Login User :

POST /api/users/login

Description: Authenticate a user and return a JWT token.

3.Get User Profile :

GET /api/users/profile

Description: Retrieve the profile information of the authenticated user.

4.Update User Profile :

PUT /api/users/profile

Description: Update the profile information of the authenticated user.

5.Get Leaderboard :

GET /api/leaderboard

Description: Retrieve the leaderboard showing top performers

→Problem Endpoints

1.Get All Problems :

GET /api/problems?page=2&limit=20

Description: Retrieve a list of all coding problems.

2.Get Problem By ID :

GET /api/problems/:id?=lang

Description: Retrieve details of a specific problem by its ID and lang.

3.Endpoint: Get Problems By Topic

GET /api/problems/topic/:topic

Description: Retrieve a list of coding problems based on a specific topic.

→Submission Endpoints

1.Submit Solution :

POST /api/submit

Description: Submit a solution for a specific problem.

2.Get User Submissions :

GET /api/submissions/userId

Description: Retrieve all submissions made by the authenticated user.

3.Get Submissions By Problem :

GET /api/submissions/problem/:id

Description: Retrieve all submissions for a specific problem.

4.Get Submission By ID :

GET /api/submissions/:id

Description: Retrieve details of a specific submission by its ID.

Basic process of submitting a solution:

1.Users browse and select coding problems from the platform,then user writes code (C++, Java, Python) in the integrated editor.
2. A Docker container is dynamically created to execute the user's code securely.It is isolated from the main server environment and provisioned with necessary dependencies.
3. The user's code is sent to the Docker container for compilation (if needed) and execution. The container runs the code and produces output (stdout) and error messages (stderr).

→This structured approach ensures secure and efficient execution of user-submitted code while maintaining platform integrity and performance. Using Docker containers enhances security by isolating code execution and enables scalable management of computing resources.

Docker Setup:

Create Dockerfile: Write a Dockerfile that defines the steps to build your application environment. This includes specifying the base

image, installing dependencies, copying application code, and configuring runtime settings.

Build Docker Image: Use docker build command to build a Docker image from the Dockerfile. This process creates a reusable, self-contained package containing everything needed to run your application.

Run Docker Containers: Use docker run command to start Docker containers based on your Docker images. Specify port mappings, environment variables, and other runtime configurations as needed.

Docker Compose: For multi-container applications, use Docker Compose to define and manage multi-service applications. Docker Compose uses a YAML file (docker-compose.yml) to configure application services and their dependencies. Use docker push and docker pull commands to upload and download

Dynamic Horizontal scaling:Horizontal scaling, also known as scaling out, refers to the strategy of increasing system capacity by adding more machines or nodes to a network rather than upgrading individual machines. This approach is particularly effective in distributed systems and cloud environments where workloads can be distributed across multiple instances.It is a key strategy for designing scalable, resilient, and responsive

systems that can efficiently handle growing user bases and fluctuating workloads in today's dynamic digital environments. It aligns with modern cloud-native principles and enables organizations to achieve high availability, performance, and user satisfaction across diverse use cases and scenarios.