Status In progress

Online Judge

Engineer Deekshith A

Introduction

Problem statement

Competitive programming platforms allow users to solve coding problems and evaluate their solutions in a real-time environment. A key requirement of such platforms is the ability to execute user-submitted code securely while ensuring fairness and scalability.An online judge is a platform that hosts these coding challenges, providing the infrastructure to manage and execute the competitions.

Objective

The goal is to design and develop an online judge platform that enables users to:

1. **Solve Coding Problems** – Users should be able to browse, attempt, and submit solutions to programming challenges.
2. **Code Execution & Evaluation** – Submitted code should be executed in a secure, isolated environment and evaluated against predefined test cases.
3. **Submission History** – Users should be able to view their past submissions along with timestamps, execution status, and code versions.
4. **Problem Contribution & Moderation** – Users can propose new coding problems, but they must be reviewed and approved by an admin before being published.
5. **Scalability & Performance** – The system should handle concurrent code submissions efficiently without blocking execution. A task queue mechanism should manage submissions asynchronously.

Requirements

Designing a Full Stack Online Judge Using Mern Stack. Takes code from different users over the server. Evaluates it automatically as accepted or not accepted

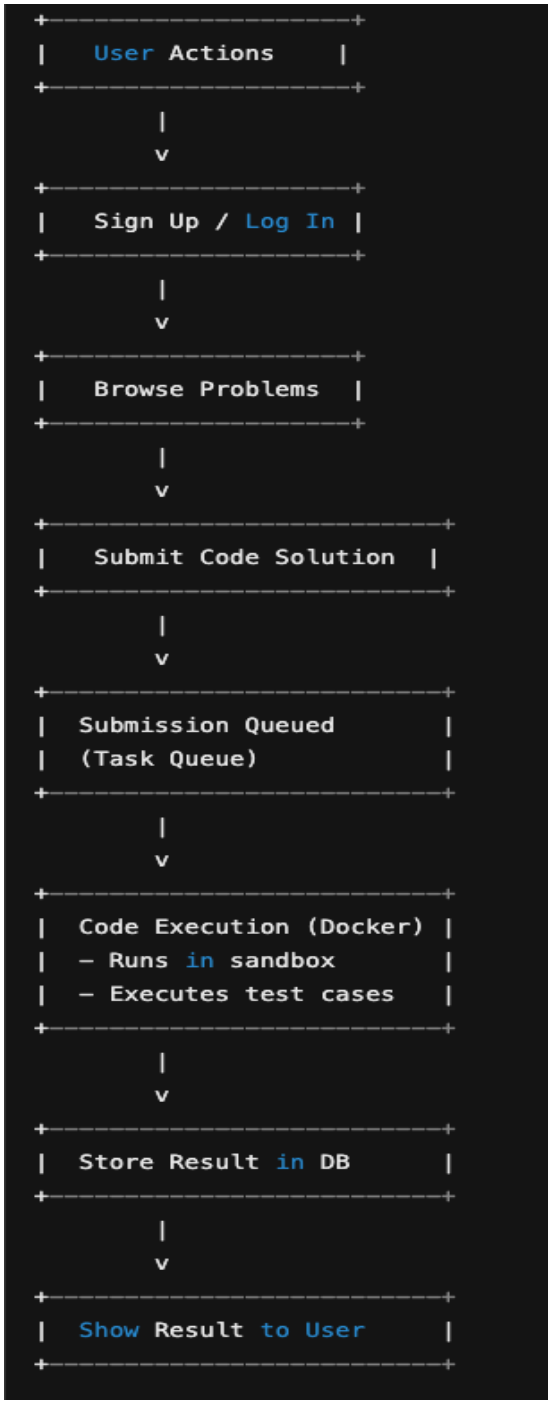
Functional Requirements

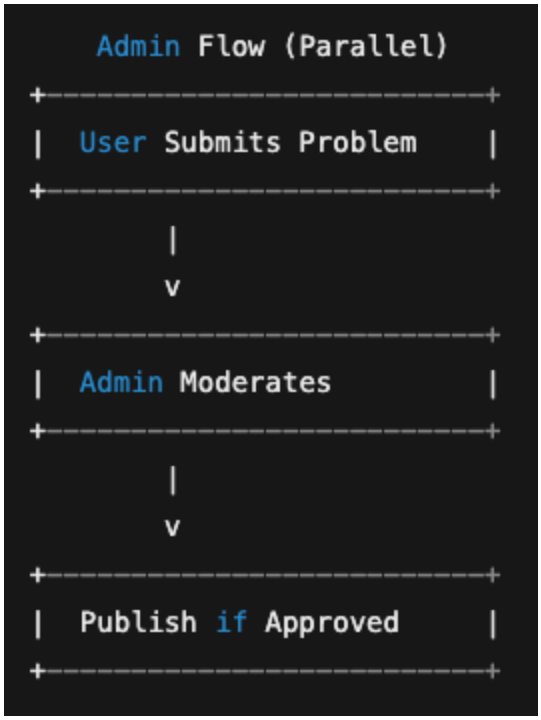
1. **User Management**
    - **Sign Up/Login:** Users can register and authenticate.
    - **User Dashboard:** Post-login view for problem solving and submission history.
2. **Problem Solving**
    - **Problem Listing:** Users can view a list of available problems.
    - **Code Submission:** Users can write, run, and submit solutions.
    - **Real-Time Feedback:** Upon submission, the system evaluates the solution against predefined test cases and returns pass/fail results.
3. **Submission History**
    - **Record Keeping:** Maintain a log of user submissions along with the code and status at the time of submission.
4. **Question Submission & Moderation**

    - **Problem Proposal:** Any user can propose new questions and provide corresponding test cases.
    - **Admin Review:** Submitted problems require approval by an admin before being published on the platform.
5. **Supported Languages**
    - Ruby, C++, Java , Python

Non Functional Requirements

1. **Performance & Scalability**
    - **Queuing Mechanism:** Implement a queuing system to handle concurrent submissions. This ensures that each code evaluation task is scheduled asynchronously, preventing blocking.
2. **Reliability & Availability**
    - **Failover:** Plan for a failover strategy for critical components such as the code evaluation service.


User Actions
Sign Up / Log In
Browse Problems
Submit Code Solution
Submission Queued
(Task Queue)
Code Execution (Docker)
– Runs in sandbox
– Executes test cases
Store Result in DB
Show Result to User


Admin Flow (Parallel)
User Submits Problem
Admin Moderates
Publish if Approved

Challenges And Solutions

1. **Performance Bottlenecks**
    - Usage of **asynchronous processing for code evaluation**, scale horizontally if necessary, and optimize database queries.
    - Possibility of **offloading heavy computation into worker nodes** as even though asynchronous nature of javascript helps us to some extent but can also block the event loop and degrade the performance for the users
2. **Security Risks**
    - Make use of Docker images and keep them secure, regularly update dependencies, enforce strict resource limitations, and monitor container activity.
3. **Database Consistency & Transactions:**
    - Transaction management especially for critical operations like recording submissions and updates.
4. **Container Security**
    - **Resource Limits**: Enforce CPU, memory, and execution time limits to prevent abuse.
    - **Isolation:** Ensure containers are isolated from each other and the host system.

- **Network Restrictions:** Limit network access within containers to reduce potential attack vectors.
- **Secure Images:** Use minimal and secure base images, keeping them up-to-date with security patches.

Architecture & Technology Stack

System Architecture

Monolithic Architecture: The entire application (frontend, backend, and code evaluation service management) is built as a monolith initially, simplifying the development for the MVP.

Backend & Frontend

Backend: Node.js using Express.

Frontend: React for a dynamic and responsive user interface.

Database

MongoDB: To store user data, problem statements, submissions, and logs.

DataBase Design

User Collection: Stores user's personal information

```
_id: ObjectId (Primary Key)
name: string (CharField)
email: string (EmailField, unique)
password_hash: string (CharField)
role: string (CharField, enum: ["user", "admin"])
created_at: datetime (DateTimeField)
```

Problems Collection: Stores problems used in platform

```
_id: ObjectId (Primary Key)
title: string (TextField, unique)
description: string (TextField)
difficulty: string(CharField, enum: ["easy", "medium",
"hard"])
constraints: list[string] (ArrayField)
tags: list[string] (ArrayField, optional)
created_by: ObjectId (ForeignKey to Users)
reviewed_by: ObjectId (ForeignKey to Admin, optional)
created_at: datetime (DateTimeField)
```

```
state: string(CharField, enum: ["pending", "approved",
"rejected", "archived"])
```

Test case Collection: Stores test cases for all the problems

```
_id: ObjectId (Primary Key)
problem_id: ObjectId (ForeignKey to Problems)
input: string (TextField)
expected_output: string (TextField)
is_sample: boolean (BooleanField, default=False)
created_at: datetime (DateTimeField)
```

Test case explanations Collection: Stores explanations for test cases if required

```
_id: ObjectId (Primary Key)
test_case_id: ObjectId (ForeignKey to Test Cases)
explanation: string (TextField)
created_at: datetime (DateTimeField)
```

Submission Collection: Stores user submissions for problems.

```
_id: ObjectId (Primary Key)
user_id: ObjectId (ForeignKey to Users)
problem_id: ObjectId (ForeignKey to Problems)
language: string (CharField, enum: ["python", "cpp",
"java", "ruby"])
code: string (TextField)
```

```
status: string (CharField, enum: ["pending", "running",
"accepted", "wrong answer", "runtime error", "time limit
exceeded"])
execution_time: float (FloatField, optional)
memory_used: int (IntegerField, optional)
submitted_at: datetime (DateTimeField)
```

Web Pages

Authentication Pages

Login Page

- Users enter email/password to log in
- OAuth options (optional)

Signup Page

- Form for new users to register

Core User Pages

Home / Dashboard

- Shows a list of problems
- Filters by difficulty, tags, etc.
- User submission stats

Problem Details Page

- Displays problem statement, constraints, sample I/O
- Code editor to write and submit solutions
- Run and Submit buttons
- Shows past submissions for this problem

Submissions Page

- List of past submissions
- Status (Pending, Accepted, Wrong Answer, etc.)
- Clicking a submission shows:
    - Submitted code
    - Execution result
    - Test cases passed/failed

Problem Management (For Users & Admins)

Create New Problem Page

- Users can submit new problems
- Fields: Title, Statement, Constraints, Test Cases
- Goes into "in_review" state

Admin Moderation Page

- Lists problems pending review
- Approve/Reject buttons
- Add comments if needed

Additional Screens

User Profile Page

- Shows username, solved problems count, ranking (In Future)
- Change password option

Leaderboard Page (Future)

- Ranks users based on number of problems solved

Settings Page (Future)

- Change email, notification settings, etc.