Byte Battles - An Online Judge

Problem Statement:

Create an online platform where a user will have their own profile, they can solve a problem in a specific coding language and run their code against custom inputs, and they can submit their code to get a verdict against pre-defined testcases. A user can also participate in a contest that is either hosted by the platform or a third party. Like that a user which can also host their contest on the platform.

Overview:

Designing a secure and scalable full stack online judge Using Django, React and MySQL.

Features

Here are the features that will be present in Byte Battles: -

User Registration: Users will be able to create an id on the platform by signing up and can create a username and password for login purposes. Also, with this username they will be able to register for contests.

Problem Set: Anyone who visits the website will be able to see the problems that are present on the platform and the features it offers, however to use these features they should have an account.

Practice Problems: A registered user will be able to open a problem code their solution for that problem in the provided code editor and run it against a custom input or submit to check their code against some testcases. If their solution is correct, they will get an “Accepted” verdict else they will get a “Rejected” verdict. There will be a “Add Notes” button also for the Journal

Contests: There will be a separate section for contests happening on the platform with their details like host, prizes, difficulty etc. Thereby in this section there will be a button to host a contest on the platform.

Online Compiler: The platform will have its very own online compiler for JAVA in a separate screen. Anyone with or without an account will be able to use this online compiler to run their code. They can enter the inputs if needed and the compiler will compile the code and give an output.

Journal: Users can also access their own private journal that they can create for each problem present on the platform with any notes they want. They can make notes about the first approach they took and then how they optimised it or the mistakes they made etc. This feature can be very helpful for revision practices.

Challenges:

1. **High traffic:** Thousands of users submitting their code at the same time

2. **Malicious Code/Script Injection:** An individual could execute harmful scripts or code within the compiler, resulting in various adverse outcomes, including some that pose significant security risks.

3. **Admin Access:** An unauthorized user gains control to alter the decisions and results on the server.

4. **User Authentication:** Risks associated with weak password policies and account security could lead to unauthorised access.

5. **Data Privacy:** Ensuring the confidentiality of the data, especially in journals and during contest participations.

6. **Server Security:** Protecting the server form DDoS attacks, unauthorised access and data breaches.

7. **Output Manipulation:** Risks of tampering with the verdicts and outputs by unauthorized individuals.

8. **Contest Integrity:** Ensuring the fairness and integrity of contests against cheating or result manipulation.

9. **Infinite loop**: The code may keep running if there is some error with it and exhaust all the resources in the process.

Solutions:

1. **High Traffic:** Implementing load balancing, using a content delivery network (CDN) and optimizing database queries to handle high volumes of traffic efficiently.

2. **Malicious Code/Script Injection:** Using sandboxing techniques or docker containers to isolate the execution environment and regularly updating compiler security.

3. **Admin Access:** Implementing multi-factor authentication, restrict access with role-based permission and regularly audit admin activities.

4. **User Authentication:** Enforcing strong password policies, offering multi-factor authentication and educating users on account security.

5. **Data Privacy:** Encrypting sensitive data and conducting regular security audits to ensure data confidentiality.

6. **Output Manipulation:** Implementing checksums to verify the integrity of outputs and verdicts.

7. **Contest Integrity:** Using randomized problem sets and implement anti-cheating measures during contests like plagiarism check etc.

8. **Put a clock:** Each problem should have its own clock after submission button is clicked, if the time exceeds the limit, then it should show the error.

High Level Design

1. **Database Designing:**
    - Table 1: Users
      Schemas:
      Username: string (varchar) (PRIMARY KEY)
      Name: string (varchar)
      Password: string(varchar)

Email: string (varchar)
Age: int
Institution: string (varchar)
Contest_id: int (Foreign Key)

- Table 2: Problems
  Schemas:
  Problem id: (Auto-Increment) (PRIMARY KEY)
  Description: string (varchar)
  Name: string (varchar)
  Difficulty: ENUM(‘Easy’, ‘Medium’, ‘Hard’)

- Table 3: Submissions
  Schemas
  Submission_id: (Auto-Increment) (PRIMARY KEY)
  User_id: int (Foreign Key)
  Problem_id: int (Foreign Key)
  Code: Text
  Verdict: ENUM(‘Accepted’ , ‘Rejected’)
  Submission time: Timestamp

- Table 4: Contests
  Schemas
  Contest_id: int (Auto-Increment) (PRIMARY KEY)
  Title: string (varchar)
  Host: string (varchar)
  Prizes: TEXT
  Difficulty: ENUM(‘Easy’, ‘Medium’, ‘Hard’)
  Contest_date: TIMESTAMP

- Table 5: Journal
  Schemas:
  User_id: int (Foreign Key)
  Problem_id: (Foreign Key)
  Notes: TEXT

- Table 6: Testcase
  Schemas
  Problem_id: int (Foreign Key)
  Input: TEXT
  Output: TEXT

2. Web Server Design:

Screen 1: Home Screen

Frontend Components:

- Problem List
- Login/Signup

Backend:

- Define a Django view to handle GET requests to fetch all problems from MySQL.
- API endpoint: /api/problems/

Screen 2: Specific Problem

Frontend Components:

- Problem statement
- Example testcase
- Coding Arena
- Verdict / Submission Log

Backend:

- Define a Django view to handle GET requests for problem details.
- When submitted the backend should handle a POST request that would evaluate the code submitted by fetching the testcases input file from testcase table and store the outputs of the code in another file and

then compare the output file form testcase table and the file generated. If same then “Accepted” verdict else “Rejected”

- API endpoint: /api/problems/<id>/

Screen 3: Journal

Frontend Components:

- Problem name

Backend:

- Link each problem to their notes.
- API endpoint: /api/Journal /<id>/

Screen 4: Contest

Frontend Components:

- Upcoming Contest
- Ongoing Contest
- Host your own contest

Backend:

- Define a Django view to handle GET requests for Contest details.
- API endpoint: /api/Contest

Screen 4: Individual Contest

Frontend Components:

- Contest Details
- Problems in contest
- Contest Leaderboard (Optional)

Backend:

- Define a Django view to handle GET requests for Contest details and problems.

- API endpoint: /api/Contest/<id>

Screen 5: Online Compiler

Frontend Components:

- Code editor
- Run

Backend:

- Define a Django view to handle POST requests for code to run on compiler.
- API endpoint: /api/Java-Compiler

Screen 5: Login/signup

Frontend Components:

- User Details
- Login signup button

Backend:

- Define a Django view to handle POST request.
- API endpoint: /api/login
- API endpoint: /api/signup

Screen 5: Submissions of a problem

Frontend Components:

- Submission details table sorted by time

Backend:

- Define a Django view to handle GET request to fetch last 10 submissions.

- API endpoint: /api/problems/problem-id/submissions