Problem Statement: Design an Online Judge like that of HackerRank/CodeChef/LeetCode.

Requirements and Goals of the System:

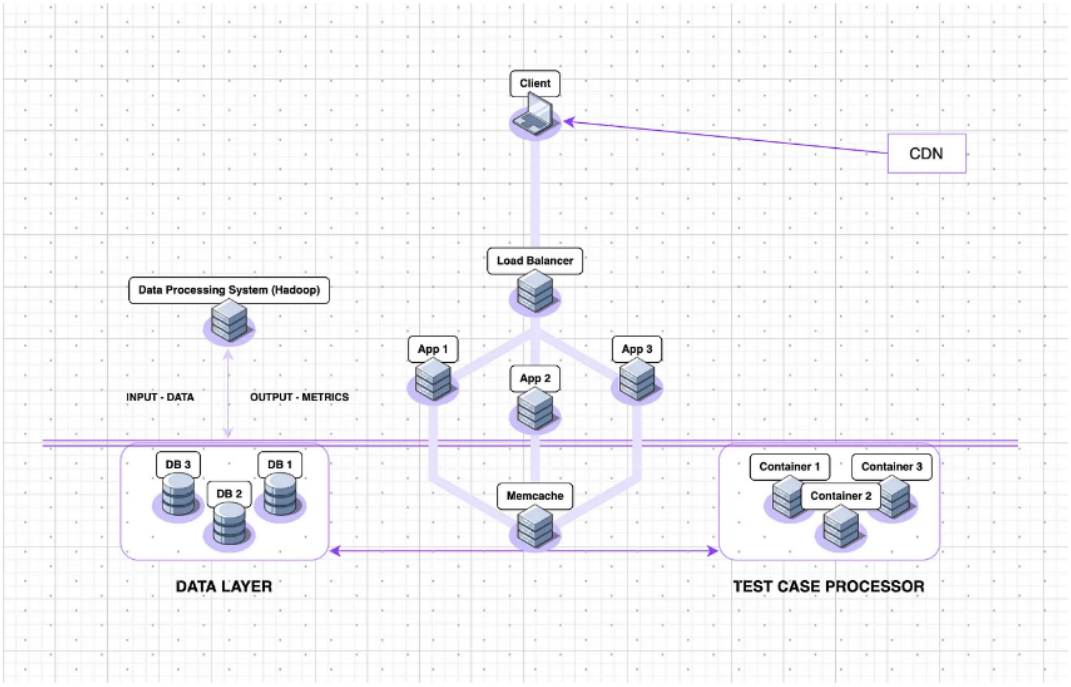
- The user should be able to submit the solution of a given problem.
- Users can participate in any competition.
- Competition should start and end in the given time frame.
- After the end time, there will be no more submissions.
- For each submission, there will be a set of test cases that should execute in the backend.
- There should be language restrictions for each given question.
- After the competition, users can see the leaderboard.
- Evolution of scores against each submission.
- Practice session for the user.
- Report generation, both in real-time and once the contest is done.

- And Many More.

Extended Requirements :

- Open Webcam and record the session.
- To detect plagiarised code, the most popular tool is the MOSS system.
- It should be horizontally scalable and easy to scale out.
- It should be highly available with an **async** design, since running code may take time.
- And Many More.

High Level Design :



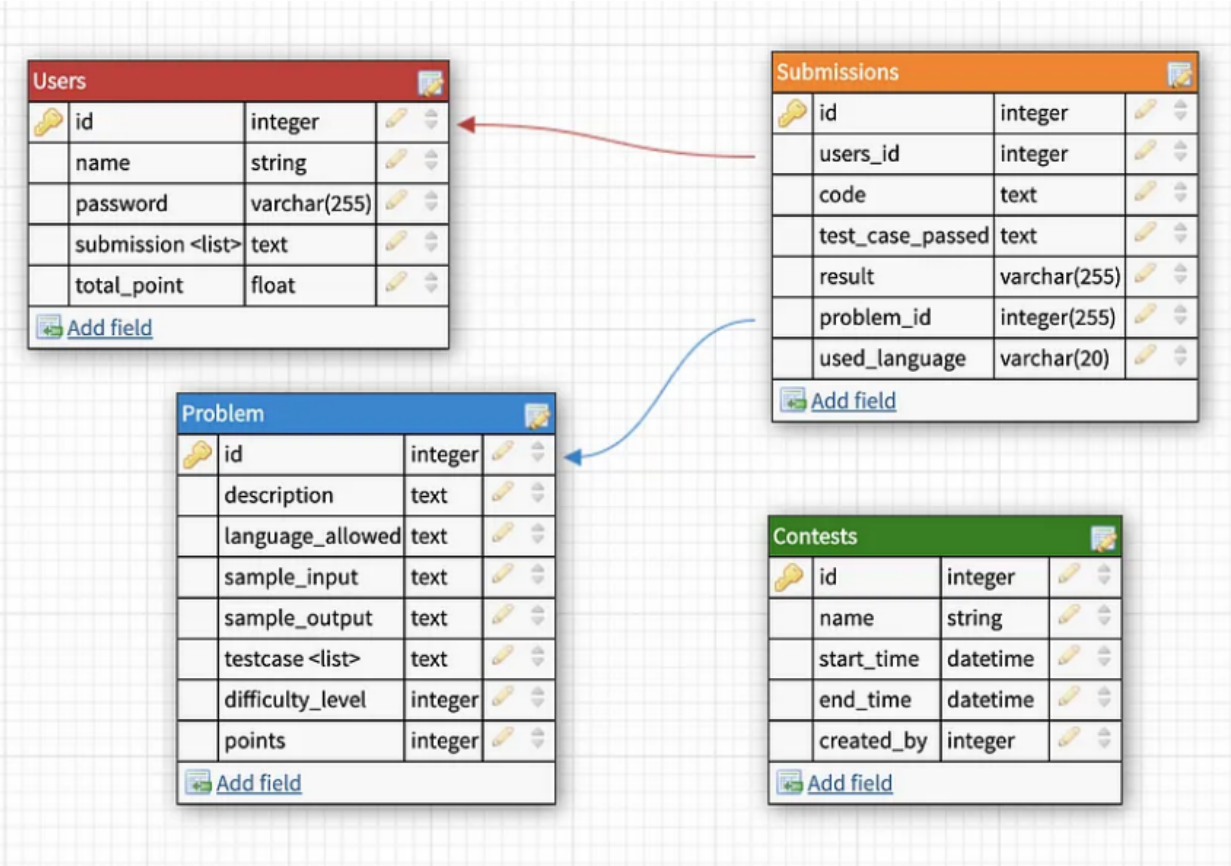
- The main communication can start using any web browser that the client is using.
- We are using load balancers to distribute the load of App servers.
- These app servers can communicate to cache servers like MemCache to fetch frequent data like **Question lists** of

particular Contests or Topics and **Test cases** for particular Questions.

- Availability is defined as the probability that the system is continuously available during a time interval and we need this most when we think of **Test Case Handler.**
- We are using distributed database here for more **Reliability and availability.**
- The Data Processing unit will process the data for metrics and report generation.
- Everything is deployed under the auto-scaling group where a set of algorithms is defined for each group regarding when to scale up and scale down.

Database Schema :

We need to store data about problems, users, their submission, their contests, leader board etc.



System APIs :

- **fetchQuestion(topic/competition)** : Fetch all question related to topic and competition. it will lookup into cache and if cache miss then it will query into DB and

return the response and at the same time it will store data in cache.

- **submitCode(code/questionId) :** submit the code to check the final output and test case result and score of particular question. it will go inside **test case processor** and return the result and same time it will update the DB fields relevant to the same.
- **runCode(code/customCase) :** It will execute our code with some given test case or some sample test case and return the response.
- **fetchSubmission(last/all) :** A user can do multiple submission at same point of time for same question, so this api will return the list for same.
- **listContest(all/city/category) :** This will return the list of contest hosted by recruiter or company itself.

- **registerForContest(user) :** This api will enrol the user to specific contest.
- **submitContest() :** This will help us to submit the all meta information of contest to DB for reporting and leader board purpose. This can be automatically called when timeout happen for specific contest and also when user completes all questions then they can submit the contest.
- **fetchLeaderBoard(contest/topic) :** This will return the user list of specific contest or topic

Component Design :

- **Load Balancing:** We can add a Load balancing layer at two places in our system “Between Clients and App servers” , “Between App servers and database replication

servers” . We can follow the Round Robin approach here. It distributes incoming requests equally among servers. This LB is simple to implement and does not introduce any overhead. Another benefit of this approach is that if a server is dead, LB will take it out of the rotation and will stop sending any traffic to it.

- **DB Replication:** A database is “slaved” to a “master” when it receives a stream of updates from the master in near real-time, functioning as a copy. The “slave” must simply apply the changes that the master validated and approved. All writes will first go to the primary server and then will be replicated to secondary servers. There are many reasons a replica makes queries return faster.
- **Cache:** The cache layer will reduce the API response time. We can store most of the read query results inside any cache. We can introduce a cache for database servers

to cache list of questions (topic wise and contest wise), Test cases (topic wise and contest wise) , Leader Board , User Information etc.

- **Test Case Processor / Executor :**

Approach 1 :

The system will test the code in an isolated environment. This way you do not have to worry about untrusted code possibly damaging your server intentionally or unintentionally. We will use Docker here to achieve the same.

Code Processor will create a new docker container with a specified language compiler/interpreter each time with some specific time like ‘n seconds’ and it will destroy itself when work is complete or time expired.

FrontEnd: Deadline: 3 Days.

- Login Screen.
- Problem List Screen.
- Contest Screen.
- Compiler Screen.
- Submission List Screen.
- Leaderboard Screen.

BackEnd: Deadline: 4 Days.

DB: 2 Days.