# OJ IIT: Open Source Online Judge Platform

**Mumuksh Tayal, Pushpendra Pratap Singh, Saatvik Rao, Sanskar Sharma**
**Indian Institute of Technology Gandhinagar, 382355, India**
**Under the guidance of Prof. Neeldhara Misra**

## I. Introduction

**Limitations of Online Judge Systems:**

**Commercial Options:** Costly for private/general use

**Educational Purposes:** Limited in scope

**Open Source:** Complex deployment for general use

**Non-Intuitive:** User-interface of state of the art OJS is hard to navigate

**Security:** Open source options may not be secure.

| Functional Requirements | Non-Functional Requirements |
|---|---|
| Well Documented API | Secure |
| Compile & Run Code | Easily Deployable |
| Sandboxed Execution | Robust / Immediacy |
| Execution Results | Configurable |
| Plagiarism Detection | Economic |

## II. System Architecture

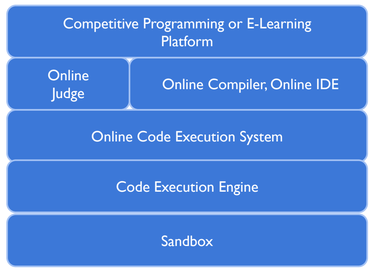



Fig: Online Judge Ecosystem Architecture

**Sandbox**: Separates programs to safely execute untrusted code with resource limits. We have used Docker Containers to achieve this.

**Code Execution Engine (CEE)**: Compiles and runs source code in a sandbox, parsing metadata.

**Online Code Execution System (OCES)**: System with a web API for compiling and running code. Uses CEE as a service and is integrated within online judge architecture.

**Online Compilers and IDEs**: Online platforms for writing, compiling, and executing code in various languages.

**Online Judges**: Online services managing submission, assessment, and scoring of code. They compile code, run it on test cases, and score based on results.

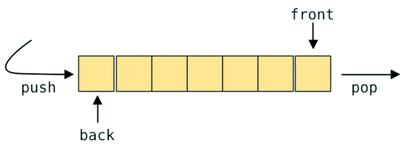



Fig: Scheduled Job Assignment using Queue

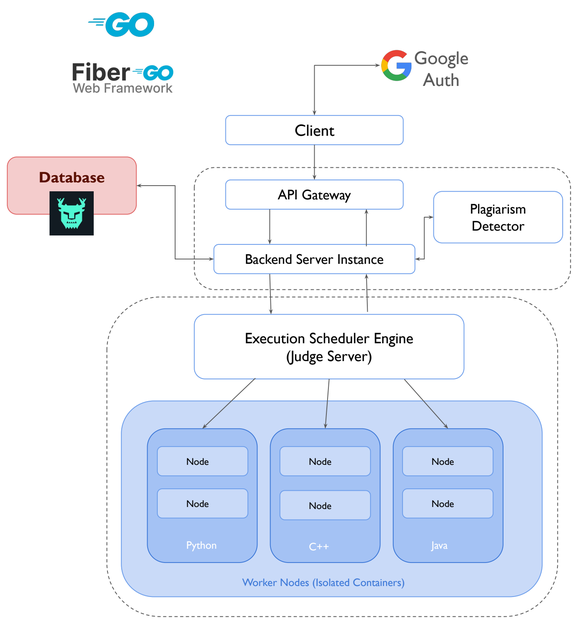



Fig: Online Judge System Architecture

**Scheduled Job Assignment:** Requests are added to queue and then assigned to separate containers by each worker thread.

**Isolated Environment Container Instantiation:** Dynamic docker container instantiation and termination handled at the judge server spontaneously.

**Channeled output responses back to the requestor:** All requests are handled independent of each other meaning collision-free execution.

**Parallel environment execution:** Each & every server instance can handle multiple users at a time, thanks to Goroutines written in Golang.

**Client**: Users sign in using their email address through Google Authentication, offering a seamless and secure login experience.

**API Gateway**: Using Go Fiber, we get a modular and easy to use web framework which ensures high performance handling of requests and routing.

**Backend Server Instance**: For every request made on application layer by user, backend server uses Go Fiber to send the appropriate request (GET, POST, UPDATE) to either frontend, database, or judge server. This architecture streamlines application operations, ensuring timely and accurate data processing across different components of the application.

**Execution Scheduler Engine (Judge Server)**: Judge server handles all the code evaluation requests coming from the users in a secure and isolated environment. It does so to avoid any unauthorized access to the server by any malicious actor.
The server also takes care of multiple requests simultaneously with the help of Goroutines coded in Golang. This means, a single backend instance can now handle multiple users at a time.
Finally, the server makes sure, none of the user requests interfere into one-another, thanks to the channelized pipeline, the user submission directly goes into a unique freshly instantiated container and the response is sent back to the backend server via the same channel..

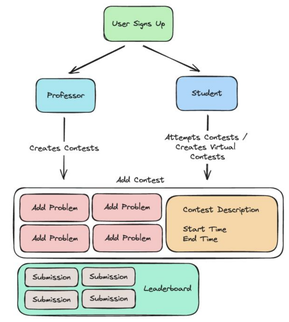
**Worker Nodes**: Each worker node controls an independent docker container which executees the user submitted code. All such node containers come with all the necessary dependencies pre-installed onto it so that the user submission can be evaluated seamlessly.

**Plagiarism Detector:** We use the tool `copydetect` as a module to identify plagiarism. It tokenizes the code and uses string matching algorithm. It generates a report for all the flagged codes.

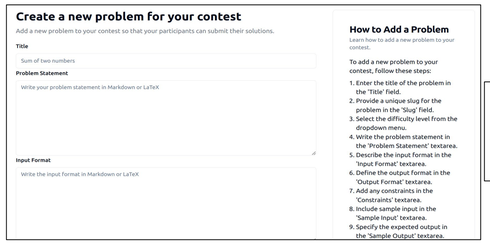
## III. Functionalities

After signing up, user can access the following functionalities

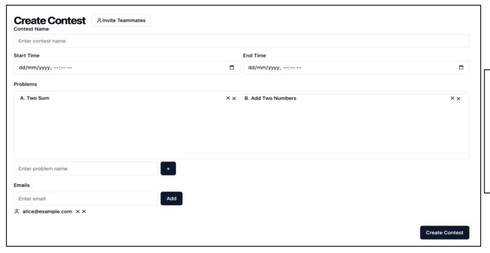
- **Add a problem:** To add problem to database
  - Problem author should specify the problem details
  - Author should specify if the problem is private or public
- **Add a test cases:** User can add test cases for a problem by writing required tests in input.txt and output.txt and assigning it to the problems.
- **Create a contest:** To create and add a contest to database
  - Contest author should select the required problems from the pool of problems they have authored and/or public problems
  - Author should specify the duration and constraint limits
  - Author can test codes for hidden test cases and plagiarism once the contest has ended
- **Run/Submit:** User can run code on open test cases. Once user submits, the code will be tested for hidden test cases after contest termination



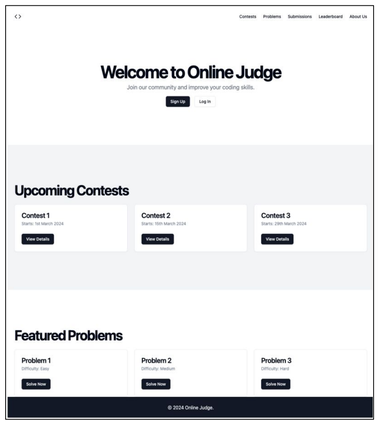

## IV. User Interface




**Create New Problems:** The Page offers a user friendly interface to create new questions for contests.




**Create Contest:** The Page allows contest authors a seamless way to add existing problems to the contest.

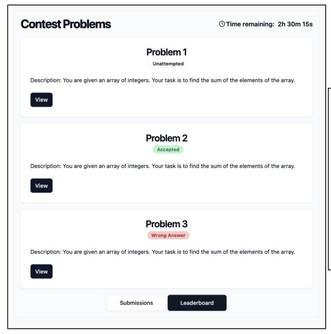



**Home Page:** The home page for the online judge web application.

User can sign-in with their Google account.

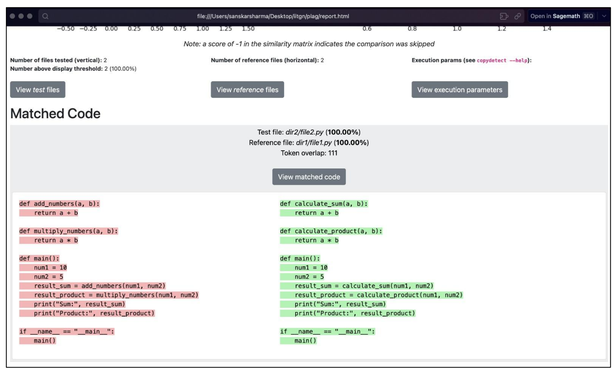
Users can only see the contests that they are eligible for.

Users can view only public problems.




**Contest Problems:** The Page shows all the problems of a selected contest in one place. The page also allows you to view all submissions and a leaderboard for that contest.

## V. Plagiarism Detection




**Plagiarism Module:**
We use the tool *Copydetect* which is a code plagiarism detection tool based on the approach called "Winnowing" and used for the popular MOSS platform.

*Copydetect* takes a list of directories containing code as input, and generates an HTML report displaying copied slices as output.

The code is broken down into smaller units (tokens) like keywords, variables, and symbols, which are then compared for similarity using string matching algorithms.\

For analysis the code may be split into sequences of N tokens (N-grams) to identify repeating patterns and similarities.