CS816 Software Production Engineering
Project Report

MicroJ - Online Judge

Submitted by

IMT2020016 Shivankar Pilligundla
IMT2020509 Hardik Khandelwal

Links

Github: https://github.com/hardik5k/MicrOJ

DockerHub:

- Frontend: https://hub.docker.com/repository/docker/shivankarp/microj_frontend
- Server: https://hub.docker.com/repository/docker/shivankarp/microj_server/general
- Judge: https://hub.docker.com/repository/docker/shivankarp/microj_judge/general

Introduction

In the ever-evolving landscape of programming education and assessment, our online judge platform emerges as a robust solution to empower both learners and educators. The platform serves as a dynamic hub where users can submit their code solutions to programming challenges, and administrators can seamlessly curate a diverse set of questions. With a focus on versatility, our online judge is designed to cater to a wide audience, from coding enthusiasts looking to hone their skills to educators seeking an efficient and automated way to assess and evaluate programming proficiency.

We have built a complete microservice architecture based application with MERN stack. Its highly focused on devops principles and standard tools for CI/CD, containerization and virtualization are used.

Devops

The successful deployment and seamless operation of our online judge platform owe much to the robust implementation of DevOps principles. DevOps, a collaborative approach that integrates development and operations teams, plays a pivotal role in ensuring the reliability, scalability, and maintainability of our system. Below, we elaborate on the key DevOps tools and practices employed in the development lifecycle of our online judge platform. Below, we elaborate on the key DevOps tools and practices employed in the development lifecycle of our online judge platform.

1. Docker and Docker Compose:
Docker provides a lightweight and efficient solution for containerization, ensuring consistent and reproducible environments across development, testing, and production. Docker Compose further facilitates the orchestration of multi-container applications, streamlining the deployment process.

2. Jenkins:
Jenkins, an open-source automation server, has been instrumental in automating various aspects of our continuous integration and continuous deployment (CI/CD) pipeline. From code integration to automated testing and deployment, Jenkins ensures a smooth and efficient development workflow.

3. Ansible:
Ansible, a powerful configuration management and automation tool, has been employed for the provisioning and configuration of our infrastructure. With Ansible playbooks, we ensure that the required dependencies and configurations are consistently applied across different environments.

4. ELK Stack:
The ELK (Elasticsearch, Logstash, Kibana) stack has been integrated to provide comprehensive log management and real-time analytics. Elasticsearch facilitates efficient log storage and retrieval, Logstash aids in log aggregation and transformation, and Kibana offers a user-friendly interface for log visualization and analysis.

5. Containers and Orchestration:
Containerization not only enhances portability but also allows for efficient resource utilization. Our use of containers, managed through tools like Docker, ensures that each component of the application is encapsulated for easy deployment and scaling. Container orchestration tools, when necessary, further automate the deployment, scaling, and management of containerized applications.

The incorporation of these DevOps tools and practices has led to a streamlined and automated development pipeline, enabling us to deliver a reliable and scalable online judge platform. This approach not only enhances the efficiency of the development process but also contributes to the overall stability and resilience of the application in diverse operational environments.

Tech Stack

Our judge is implemented using MERN stack with some additional tools like RabbitMQ, Redis Caching, etc. Runs Code submissions in a secure environment and Passes Verdicts based on pre-saved testcases. It uses RabbitMQ for Queueing the submissions, Redis for Caching the results and Docker for Sandboxing. It is a Remote Code Execution Engine, linked to an online judge.

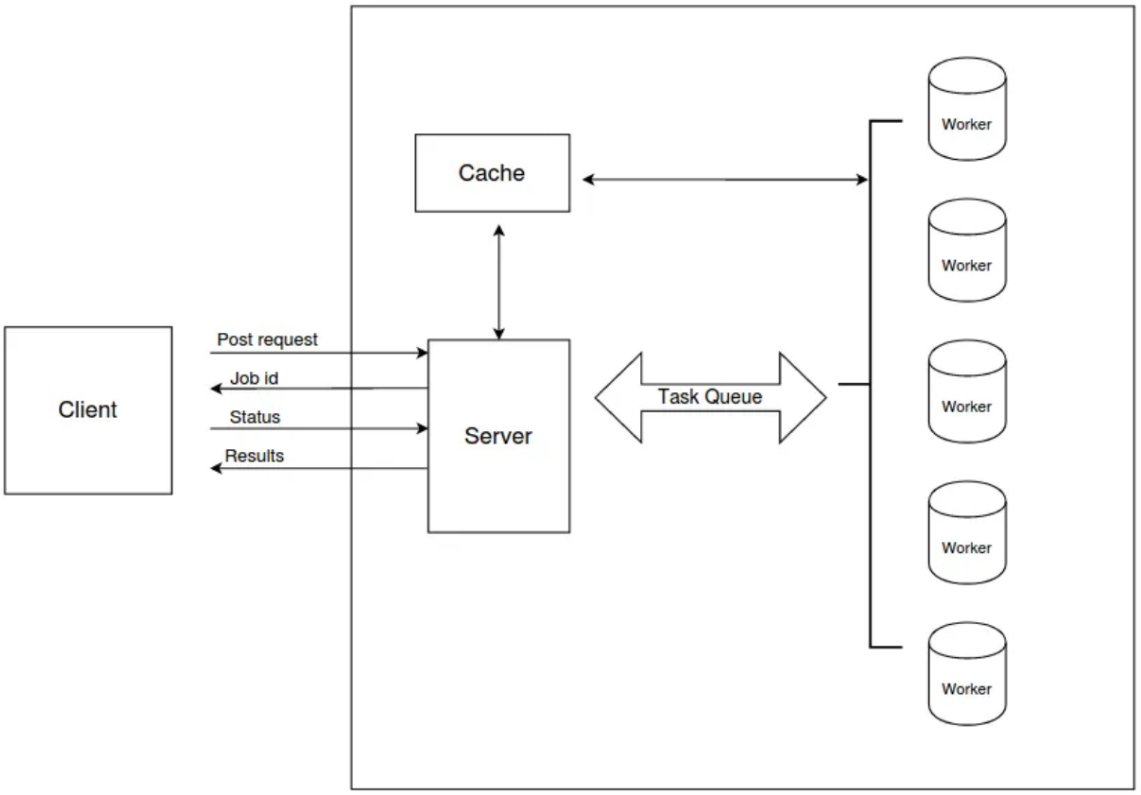
Why Message Queuing?

The Problem Lies in the scale, This kind of execution would work fine without Message Queueing with a small scale, where at a time, the API might only face about 10-15 Submission at a time at most. However For a lager Scale, it becomes Impossible to execute all of them at once. One solution could be rate Limiting, but It would be unfair in a coding contest if your submission is held because of rate limiting, and hence Message Queueing was a Legitimate option.

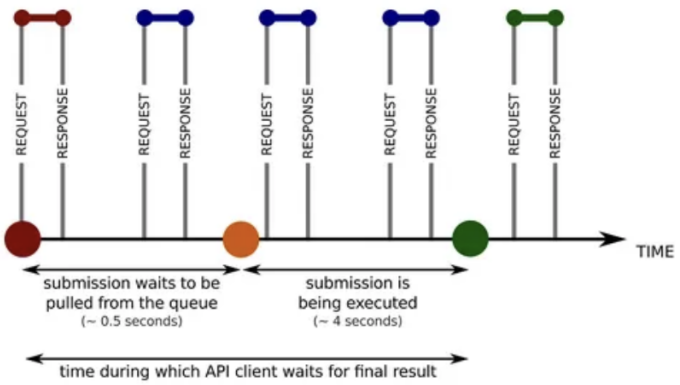
Why Docker?

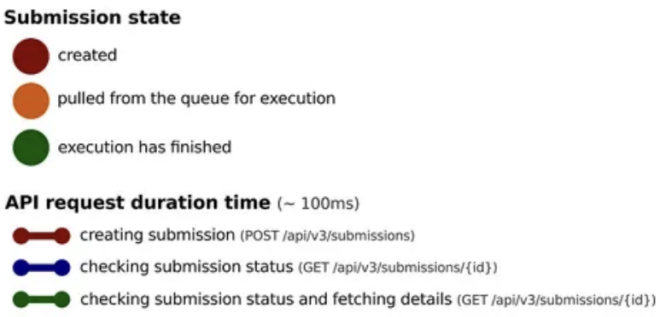
Let us Consider a case where a person tries to submit a malicious Code, like try to do a Fork attack or execute "rm-rf". In such Cases Docker Plays a crucial role in Containerizing the effects of such malicious code.

HighLevel Architecture



Submission Polling:



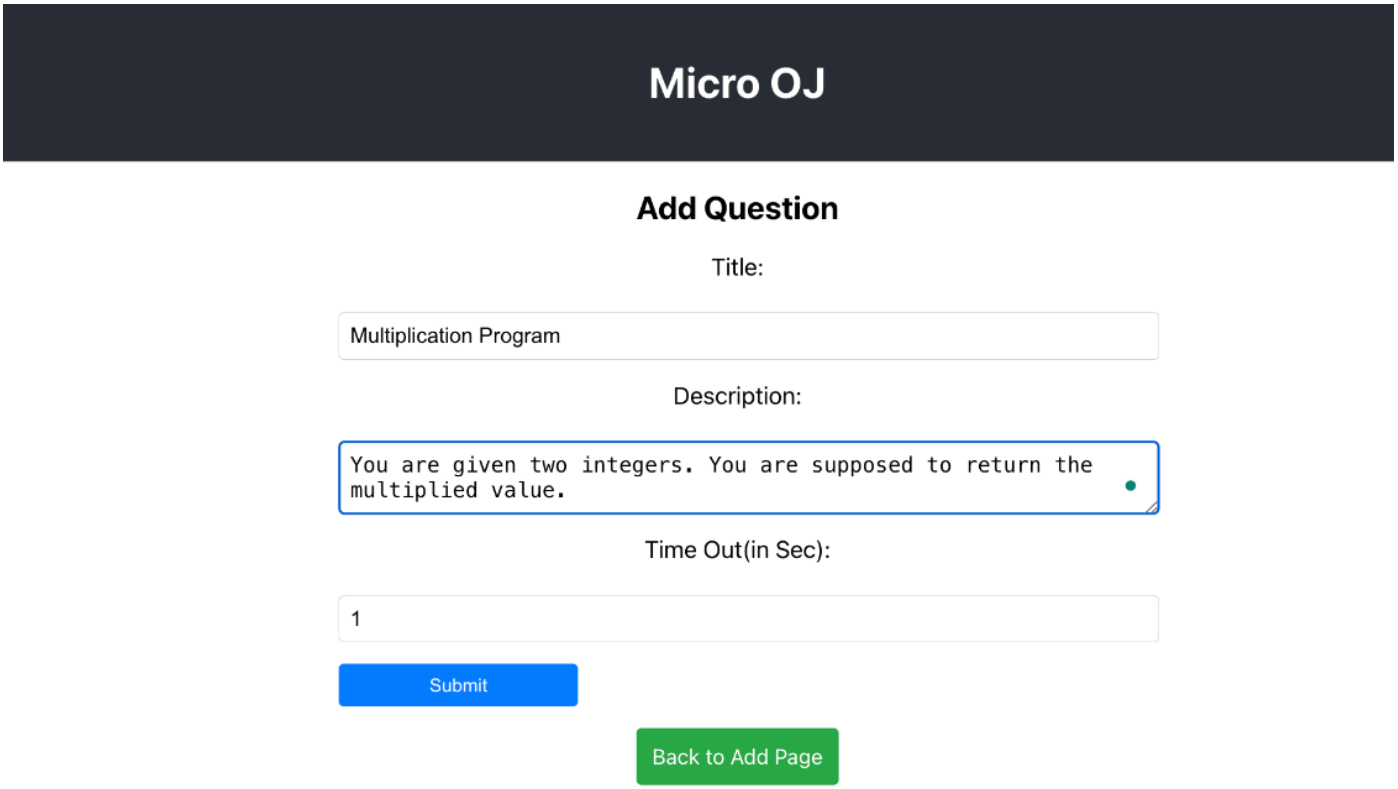


Application:

There are two major user roles. Administrator and user. Administrator can add questions, manage test cases, etc. Users can choose a question submit a solution and get their verdict for their submitted solution by the judge. The below images highlight the functionality in a high level.

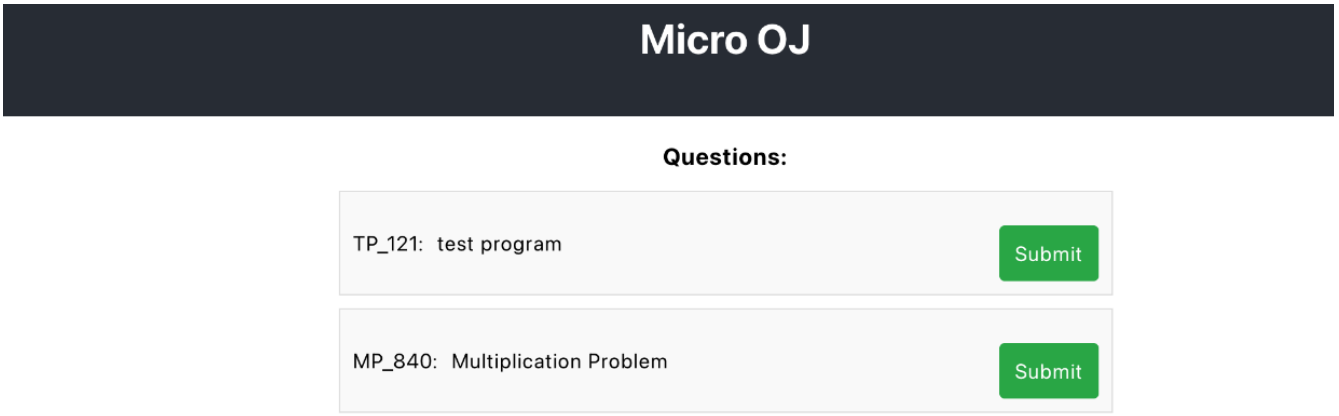
Add Question:

User will give a title, description - which has problem statement input output format etc. Then there is a timeout field which represents execution timeout. So if a code executes for more than the mentioned timeout then TLE error will appear as the verdict.



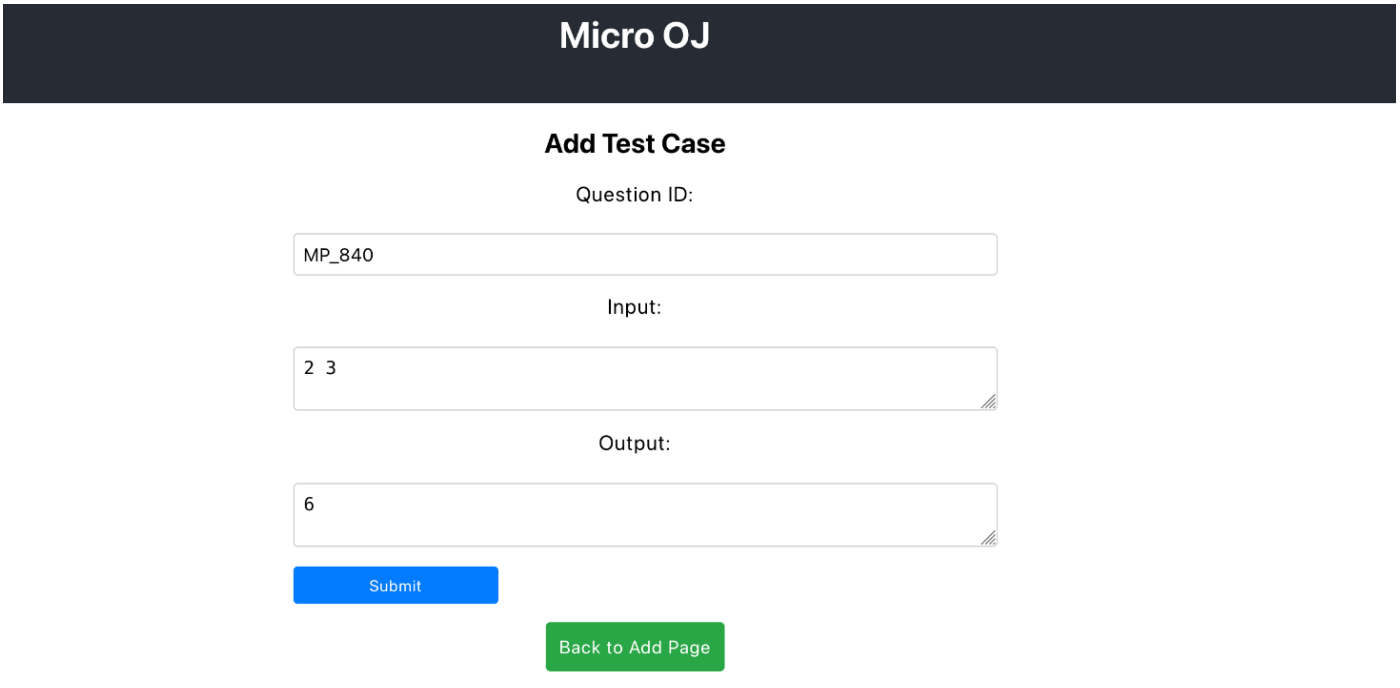
View Question

A user can view all the questions for which they can submit a solution. There will be a list of questions each with question id, title with a submit button. Clicking on submit button will redirect them to a page where code can be submitted for judging.



Add TestCase:

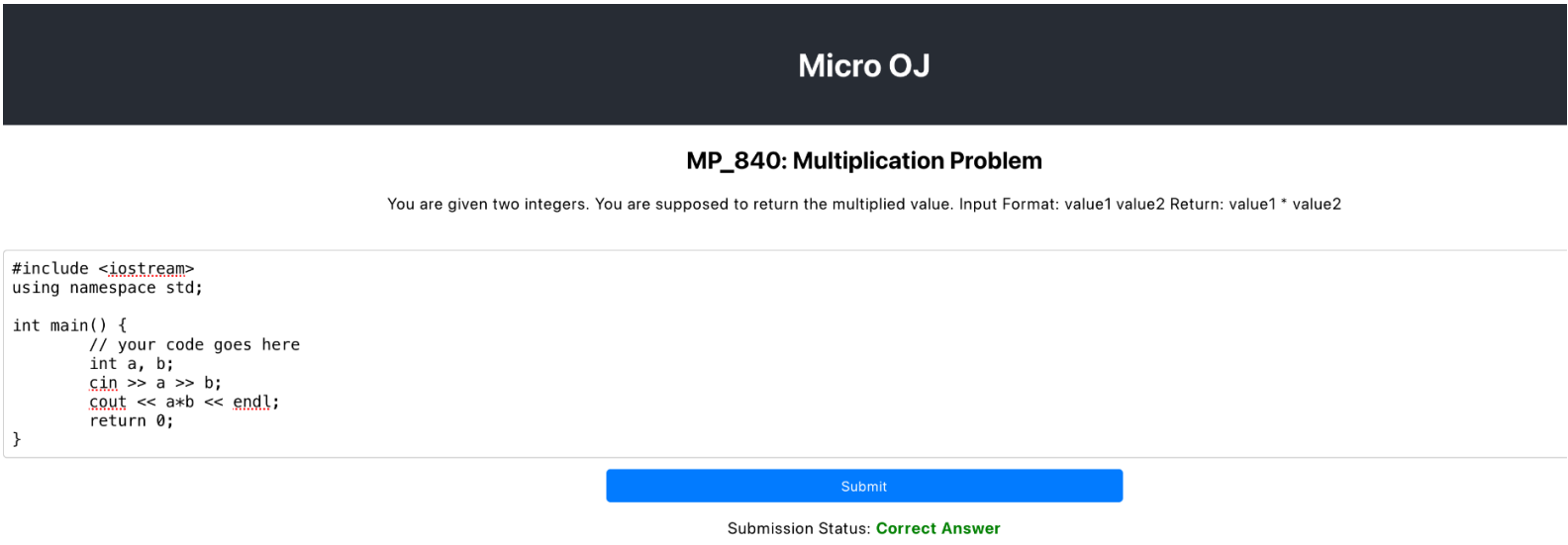
Administrator can add and manage test cases for a specific question by entering the question_id, input and expected output.



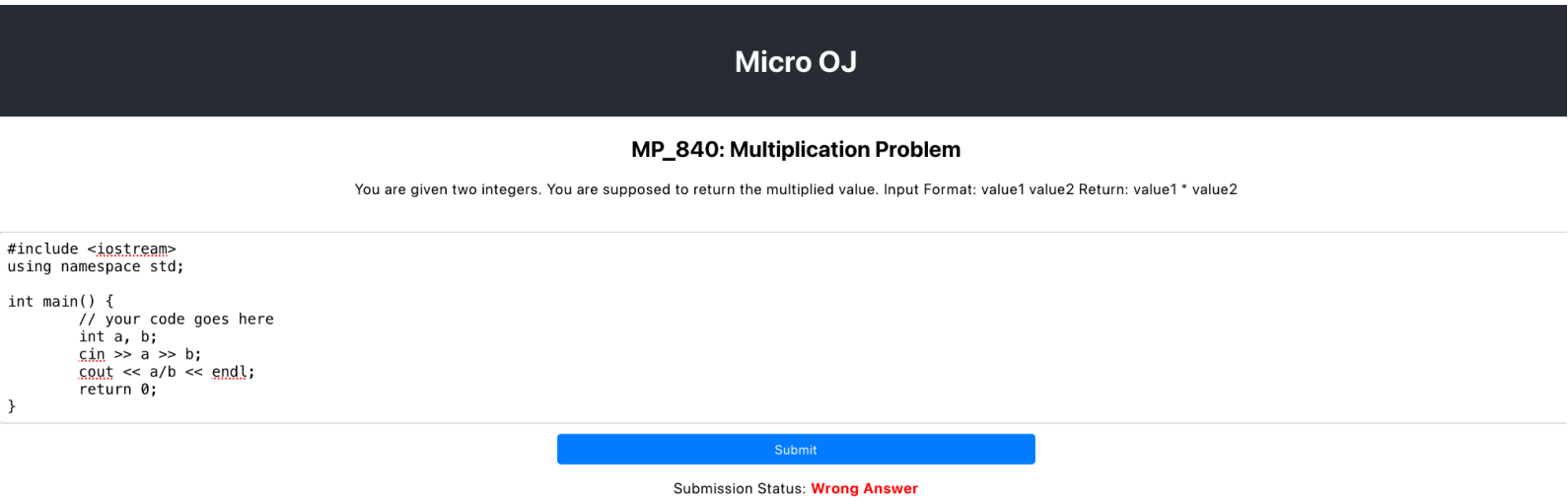
Submit Solution

The user can submit solution to a particular question and verdict will be displayed. For now only c++ language is supported. We will add implementation of other languages going further.

correct answer:

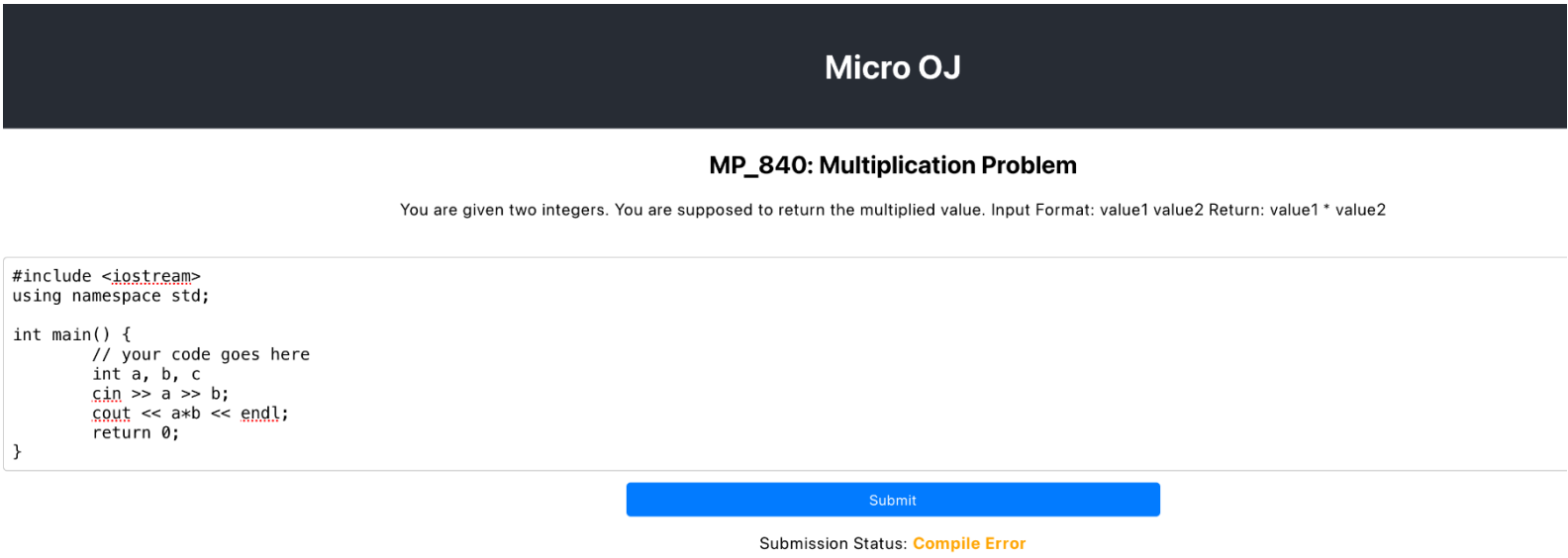


wrong answer:



Compile Error:

due to missing semi-colon



Containerization:

Frontend

```
frontend > Dockerfile
1   FROM node:18
2   WORKDIR /
3   COPY package*.json .
4   RUN npm install
5   COPY . .
6   CMD ["npm", "start"]
7
```

Server

Server > Dockerfile

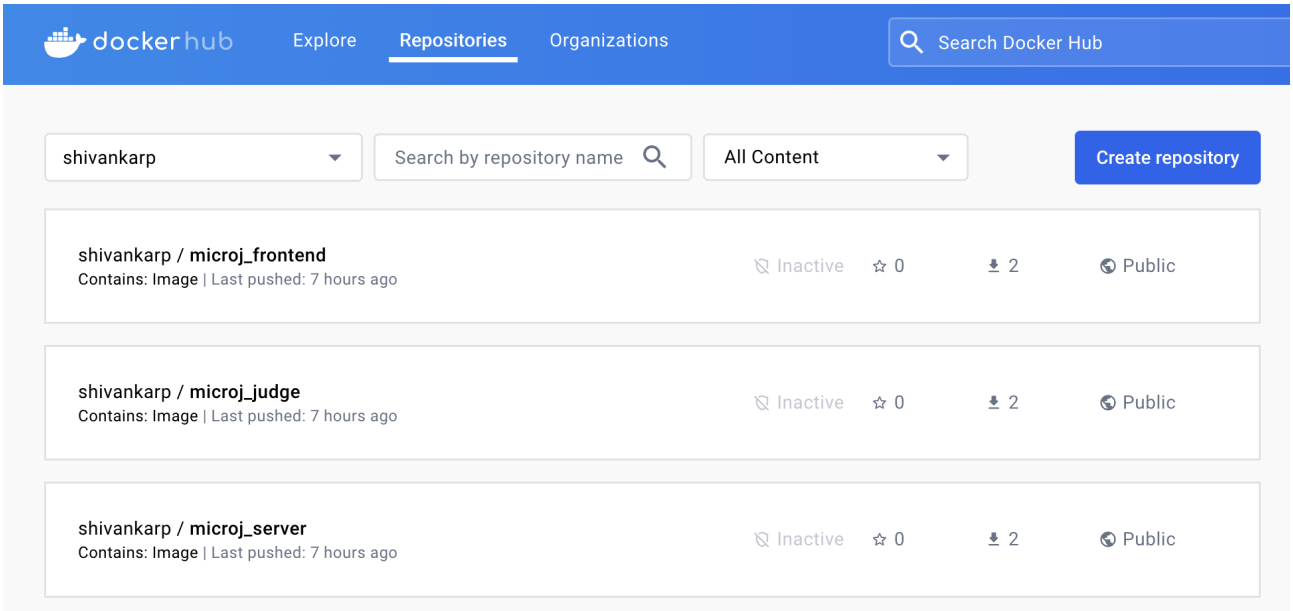
```
1   FROM node:18
2   WORKDIR /
3   COPY package*.json .
4   RUN npm i
5   COPY . .
6   CMD npm start
```

Judge

Judge > Dockerfile

```
1   FROM node:18
2   WORKDIR /
3   RUN apt-get update
4   RUN apt-get install apt-utils
5   RUN apt-get install gcc -y
6   RUN apt-get install g++ -y
7   COPY package*.json .
8   RUN npm i
9   COPY . .
10  CMD npm start
```

DockerHub:



Logging:

We have implemented logging using winston logger for node applications.

```
// Define the file transport options
const fileOptions = {
  level: 'info', // Set the log level
  filename: path.join(logDirectory, 'app.log'), // Specify the log file name
  handleExceptions: true,
  maxsize: 10 * 1024 * 1024, // Set the maximum log file size (in bytes)
  maxFiles: 5, // Set the maximum number of log files to keep
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
};

// Create the logger with the file transport
const logger = winston.createLogger({
  transports: [
    new winston.transports.File(fileOptions),
  ],
  exitOnError: false,
});

// Add a console transport for logging to the console during development
if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console());
}

module.exports = logger;
```

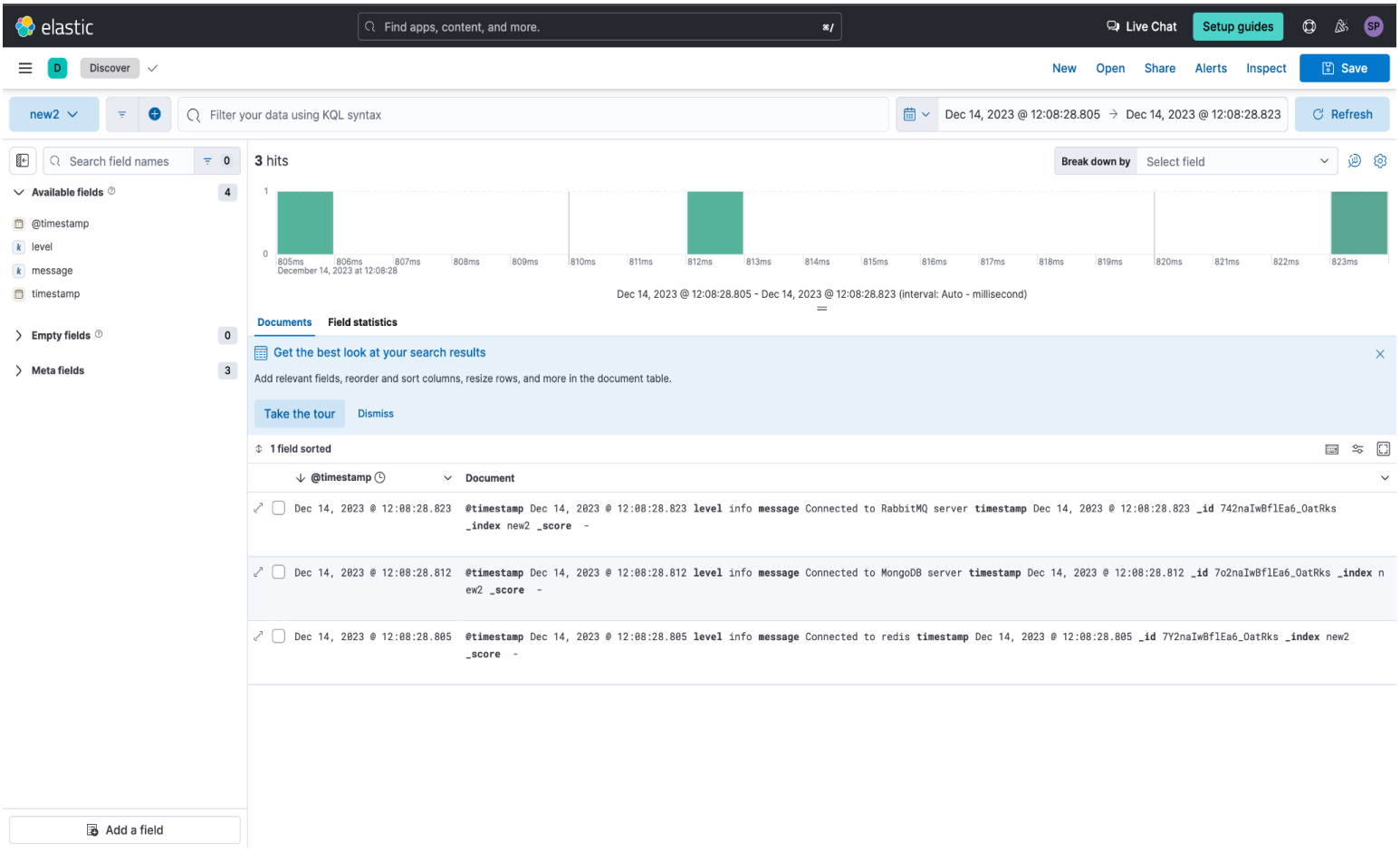
Docker Compose:

docker-compose.yml

```yaml
version: '3'
services:
  server:
    container_name: server
    restart: always
    # image: shivankarp/microj_server:latest
    build: frontend
    ports:
      - '3000:3000'

  judge:
    container_name: judge
    restart: always
    # image: shivankarp/microj_judge:latest
    build: judge

  mongo:
    container_name: mongo_service
    image: 'mongo:7'
    expose:
      - 27017

  redis:
    container_name: redis_service
    image: 'redis:4.0.11'
    expose:
      - 6379

  rabbitmq:
    container_name: queue_service
    image: rabbitmq:management
    expose:
      - 5672
    ports:
      - '15672:15672'

  frontend:
    container_name: frontend
    restart: always
    # image: shivankarp/microj_frontend:latest
    build: frontend
    ports:
      - '4000:3000'
    depends_on:
      - server

```

ELK Stack:

Kibana Dashboard



Continuous Integration:

We have used jenkins for CI/CD pipeline and integrated with github SCM polling to track new commits and trigger a build. once new builds are done then they are pushed to dockerhub. Then docker compose is ran using ansible.

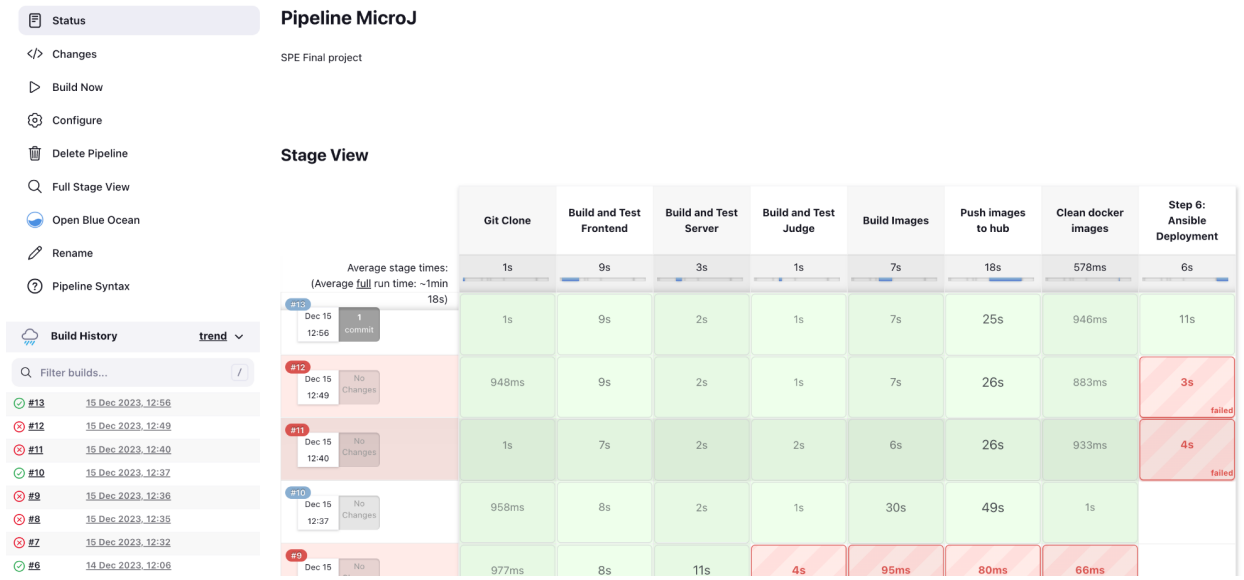
pipeline script

```
pipeline {
    agent any

    environment {
        FRONTEND_IMAGE_TAG = 'v1.0.0'
        SERVER_IMAGE_TAG = 'v1.0.0'
        JUDGE_IMAGE_TAG = 'v1.0.0'
    }

    stages {
        stage('Git Clone'){
            steps {
                git branch: 'main',
                url:'https://github.com/hardik5k/MicrOJ'
            }
        }

        stage('Build and Test Frontend') {
            steps {
                dir('frontend') {
                    script {
                        sh 'npm install && npm run build && npm test'
                    }
                }
            }
        }

        stage('Build and Test Server') {
            steps {
                dir('Server') {
                    script {
                        sh 'npm install && npm test'
                    }
                }
            }
        }

        stage('Build and Test Judge') {
```

```
        steps {
            dir('Judge') {
                script {
                    sh 'npm install && npm test'
                }
            }
        }
    }


    stage('Build Images') {
        steps {
            dir('Server') {
                script{
                    docker_image_server = docker.build "shivankarp/microj_server:latest"
                }

            }
            dir('Judge') {
                script{
                    docker_image_judge = docker.build "shivankarp/microj_judge:latest"
                }
            }
            dir('frontend') {
                script{
                    docker_image_frontend = docker.build "shivankarp/microj_frontend:latest"
                }
            }

        }
    }
    stage('Push images to hub'){
        steps{
            script{
                docker.withRegistry('', "DockerhubCred"){
                    docker_image_server.push()
                    docker_image_judge.push()
                    docker_image_frontend.push()
                }
            }
        }
    }
    stage('Clean docker images'){
```

```
        steps{
            script{
                sh "docker container prune -f"
                sh "docker image prune -f"
            }
        }
    }
    stage('Ansible Deployment'){
        steps{
            ansiblePlaybook becomeUser: null,
            colorized: true,
            credentialsId: 'ssh-cred',
            disableHostKeyChecking: true,
            installation: 'Ansible',
            inventory: 'Deployment/inventory',
            playbook: 'Deployment/deploy.yml',
            sudoUser: null
        }
    }
  }
}
```

Pipeline:



Configuration Management:

We have used ansible for configuration Management.

Deployment > ! deploy.yml

```yaml
---
- name: microJ
  hosts: my_server
  become: true

  tasks:
    - name: Copy Docker Compose file
      ansible.builtin.copy:
        src: ../docker-compose.yml
        dest: /home/shivankar/docker-compose.yml
        owner: shivankar
        group: shivankar
        mode: '0644'

    - name: Ensure Docker Compose is installed
      ansible.builtin.package:
        name: docker-compose
        state: present

    - name: Run Docker Compose
      ansible.builtin.command:
        cmd: "docker-compose -f /home/shivankar/docker-compose.yml up -d"
      args:
        chdir: /home/shivankar

```

inventory

Deployment > inventory

```
[my_server]
103.156.19.244:33033 ansible_ssh_user=shivankar ansible_ssh_pass=Test@123
```

Run by using:

ansible-playbook -i inventory deploy.yml --ask-become-pass

```
~/MicrOJ/Deployment main *1 !3 ?1 > ansible-playbook -i inventory deploy.yml --ask-become-pass            11s

BECOME password:

PLAY [microJ] ***************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************
ok: [103.156.19.244]

TASK [Copy Docker Compose file] *********************************************************************************
changed: [103.156.19.244]

TASK [Ensure Docker Compose is installed] ***********************************************************************
ok: [103.156.19.244]

TASK [Run Docker Compose] ***************************************************************************************
changed: [103.156.19.244]

PLAY RECAP ******************************************************************************************************
103.156.19.244             : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignore
d=0

~/MicrOJ/Deployment main *1 !3 ?1 >                                                                        11s
```

Testing

We have used jest library for testing the application. Jest is a popular JavaScript testing framework that is widely used for testing JavaScript code, including applications, libraries, and frameworks.

```
> MicrOJ Server@1.0.0 test
> jest

 PASS  ./server.test.js
  ✓ Check is unique id is correctly generated (1 ms)
  ✓ Check if a valid test case ID returns the correct test case
  ✓ Check if an invalid test case ID returns null

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.4 s, estimated 1 s
Ran all test suites.
~/MicrOJ/Server main *1 >
```