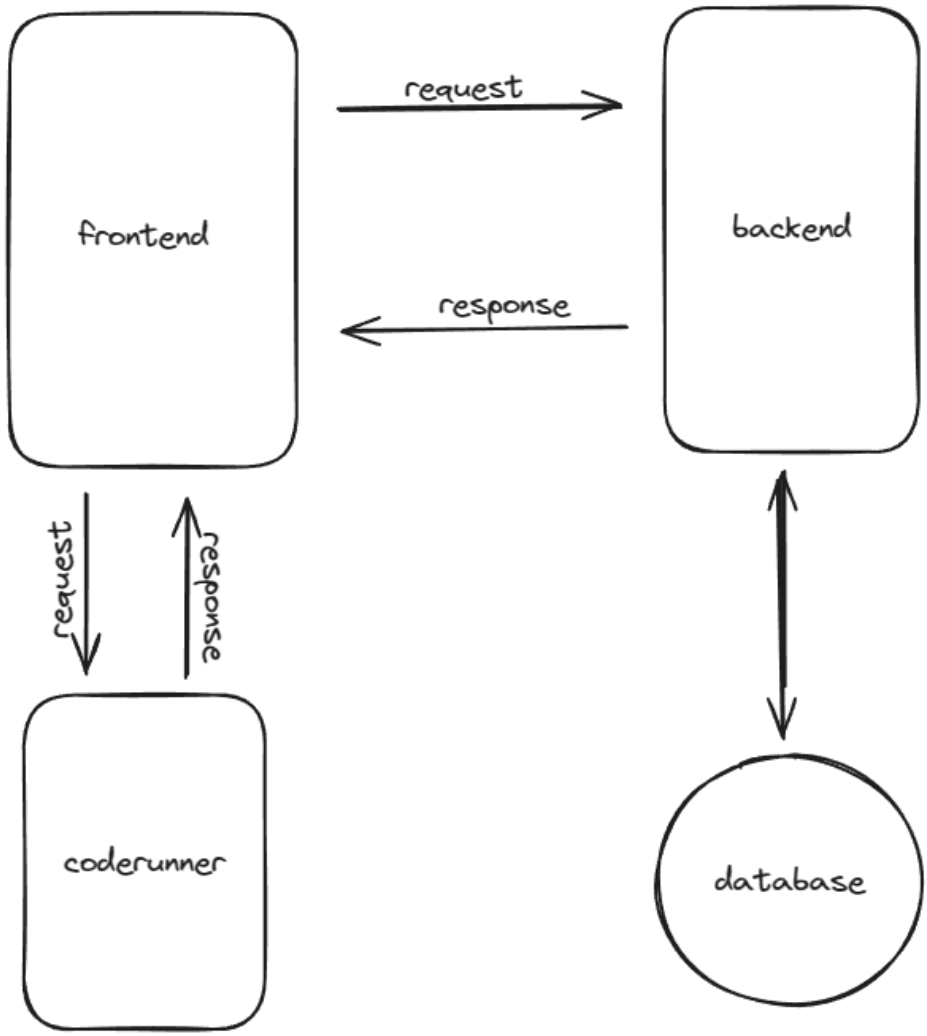
SOFTWARE PRODUCTION ENGINEERING

Online Code Editor

Chaithanya Reddy -IMT2020054
Meda Madhav - IMT2020022

INTRODUCTION

The online code editor is a web-based tool for writing, editing, saving, deleting, and executing C++ files. We used MERN (MongoDB, ExpressJS, ReactJS, NodeJS) to build this application.



Features

1. **Login/Signup**

   Account creation and login for users.

2. **Dashboard**

   Shows all User files.

3. **Add file**

   User can create new empty files.

4. **Delete file**

   User can delete any file.

5. **Code editor**

   User can open any file and edit files using the code editor.

6. **Save file**

   User can save files in the code editor.

7. **Run**

   User can execute the code.

8. **Input and Output**

   Input and output fields are present in the code editor so the user can give input and run the code, they can check the output.

DevOps Tools

1. Source Control Management: Git/Github
2. Continuous Integration: Jenkins
3. Containerization: Docker
4. Container Orchestration: Docker compose
5. Testing: React Testing Library, Jest
6. Continuous Deployment: Ansible
7. Logger: Morgan
8. Monitoring: ELK Stack

Source Control Management (SCM)

Source code management (SCM) is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. For our project, we are making use of Git for local repositories integrated with Github for remote repositories.

The project repository link: https://github.com/chaithanya99/SPEcodeeditor.git

There are 3 dedicated folders where each consists of source code, test files, dockerfile etc.

1. Front-end in “frontend/” directory
2. Back-end in “backend/” directory
3. Coderunner in “coderunner” directory

We initialize the local repository using:

- git init
- git remote add origin https://github.com/chaithanya99/SPEcodeeditor.git

Steps to push changes into the “master” branch:

- git pull origin master
- git add .
- git commit -m <Commit Message>
- git push origin master

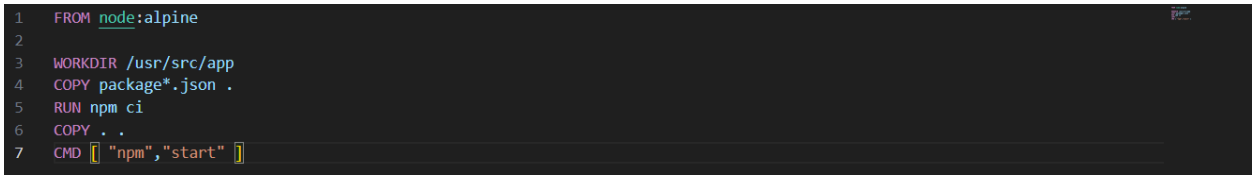
Docker

Frontend Dockerfile

```
FROM node:alpine

WORKDIR /usr/src/app
COPY package*.json .
RUN npm install
COPY . .
CMD [ "npm","start" ]
```

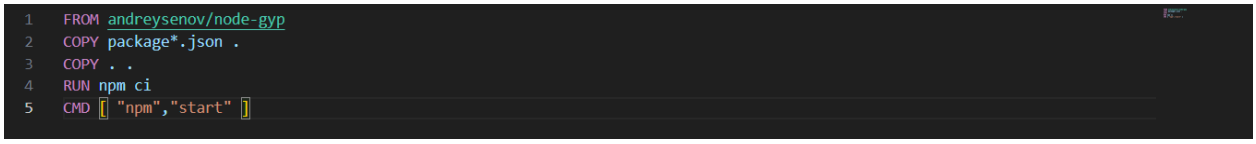
Backend Dockerfile



```
FROM node:alpine

WORKDIR /usr/src/app
COPY package*.json .
RUN npm ci
COPY . .
CMD [ "npm","start" ]
```

Coderunner Dockerfile



```
FROM andreysenov/node-gyp
COPY package*.json .
COPY . .
RUN npm ci
CMD [ "npm","start" ]
```

Running docker build command using these dockerfile creates the docker images, which will be pushed into the docker hub. This part will be covered in the Continuous Integration Section. For the frontend and backend, we are using a base image as “node:alphine”. But for the Coderunner we are using “andreysenov/node-gyp”, we need to node and g++ compiler because we are executing the code in this server, and both are installed in the andreysenov/node-gyp image.

Docker Compose

Docker Compose is a tool that was developed to help define and share multi-container applications. We can write a yml file to define the multiple services and spin multiple containers using a single command. If we have a compose file then it becomes easy for other people to run the application, they can simply clone the repo, can use a single command to start the application.

```yaml
version: '3.9'

services:
  mongo_db:
    container_name: db_container
    image: mongo:latest
    restart: always
    volumes:
      - mongo_db:/data/db

  api:
    image: chaithanya970/codeeditor:backend
    container_name: codeeditor-backend
    ports:
      - 5001:5001
    environment:
      PORT: 5001
      MONGODB: mongodb://mongo_db:27017
    depends_on:
      - mongo_db
    volumes:
      - ./access.log:/usr/src/app/access.log

  code-runner:
    image: chaithanya970/codeeditor:coderunner
    container_name: codeeditor-coderunner
    ports:
      - 5005:5005

```

We can define the list of services we want to run as part of our application. We will be running 4 services (frontend, backend, database, and coderunner). We can see in the above image we created a volume for the database container (db_container) so that data will be persistent. We can see that in the api service we mapped the port 5001 of host machine with the port 5001 of container so that frontend will be able to communicate with the backend. We did the same thing with the coderunner and frontend as well. We created a volume for the log file in the api service so that log file in the container will be synchronized with log file in the host machine.

```
29
30
31      front_end:
32        image: chaithanya970/codeeditor:frontend
33        container_name: codeeditor-front_end
34        ports:
35          - 3000:3000
36
37        depends_on:
38          - mongo_db
39          - api
40          - code-runner
41
42
43    volumes:
44      mongo_db: {}
```

This image is the continuation of the above image. We map front_end container port 3000 with the host machine port 3000. So we can access the application on port 3000 of the host machine.

Ansible

Ansible is an open source, command-line IT automation software application written in Python. It can configure systems, deploy software, and orchestrate advanced workflows to support application deployment, system updates, and more.

We are going to make use of the “Ansible” tool for the deployment stage of this project.

The files related to deployment are stored in the directory “deployment/” in github, which consists of the inventory file and an ansible-playbook for deploying the project.

The ansible-playbook for deployment namely “deployment.yaml” has the following contents.

deployment > ! deployment.yaml

```yaml
---
- name: Deploying Code-Editor application
  hosts: all
  vars:
  tasks:
    - name: Copy docker-compose file
      copy:
        src: ../docker-compose.yml
        dest: ~/Desktop/Code-Editor/

    - name: Check if log file exists
      command: ls ~/Desktop/Code-Editor/access.log
      register: file_check
      ignore_errors: yes

    - name: Create log file if not existing
      file:
        path: ~/Desktop/Code-Editor/access.log
        state: touch
      when: file_check is failed

    - name: Stop running containers
      command: docker stop codeeditor-backend db_container codeeditor-coderunner codeeditor-front_end
      ignore_errors: yes

    - name: Remove existing containers
      command: docker rm codeeditor-backend db_container codeeditor-coderunner codeeditor-front_end
      ignore_errors: yes

    - name: Docker-compose Pull
      command: docker compose pull
      args:
        chdir: ~/Desktop/Code-Editor

    - name: Prune old images
      command: docker image prune -f

    - name: Run Docker containers
      command: docker compose up -d
      args:
        chdir: ~/Desktop/Code-Editor
```

The following are the steps taken to achieve deployment:

1. We copy “docker-compose.yml” onto the managed nodes/target systems using the “copy” module. It is copied to the Desktop of the target system’s user i.e. “~/Desktop/Code-Editor”. All the files to be deployed are done here.
2. We check if the log-file for the project exists or not. If it doesn’t exist, then we create an empty file in the project folder.

3. If there exist any of the front-end, back-end, coderunner, mongo-db container of this project running, then we stop such containers and remove them using “docker stop” & “docker rm” commands.
4. Now we are ready to deploy the latest version of the project. We first pull the latest images of the project using “docker compose pull”. We are required to change the directory to “~/Desktop/Code-Editor” by passing “chdir” as an argument to the “command” module.
5. Now, we prune any dangling images that may have been created using “docker image prune -f”.
6. Finally, we create the containers for running the project using “docker compose up -d”. We make ansible run this command in the project folder by passing “chdir” as an argument to ansible. We use the “-d” flag to start the containers in the background.

Now, the target system’s user can access the project using the browser on “localhost:3000”.

Jenkins

Jenkins is an open source continuous integration/continuous delivery and deployment (CI/CD) automation software DevOps tool written in the Java programming language. It is used to implement CI/CD workflows, called pipelines.

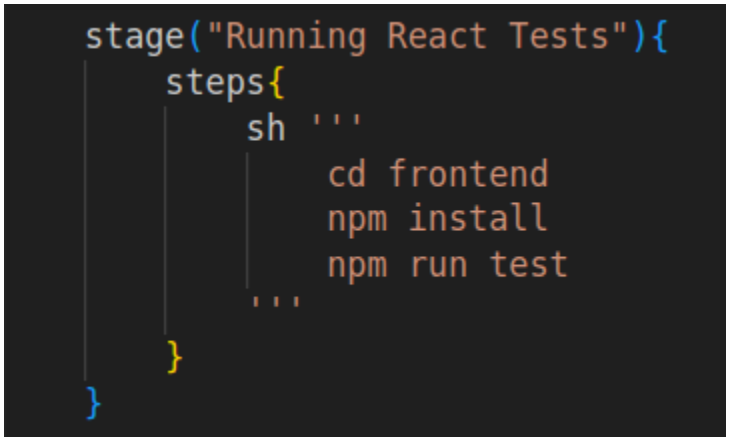
We have created a new Jenkins pipeline project. The pipeline script is written in “Jenkinsfile” that is present in the Github repository. The Jenkinsfile contents is as follows:

Stage-1:

```
pipeline{
    agent any
    stages{
        stage("Git clone"){
            steps{
                git branch: 'master',
                url:"https://github.com/chaithanya99/SPEcodeeditor.git"
            }
        }
```

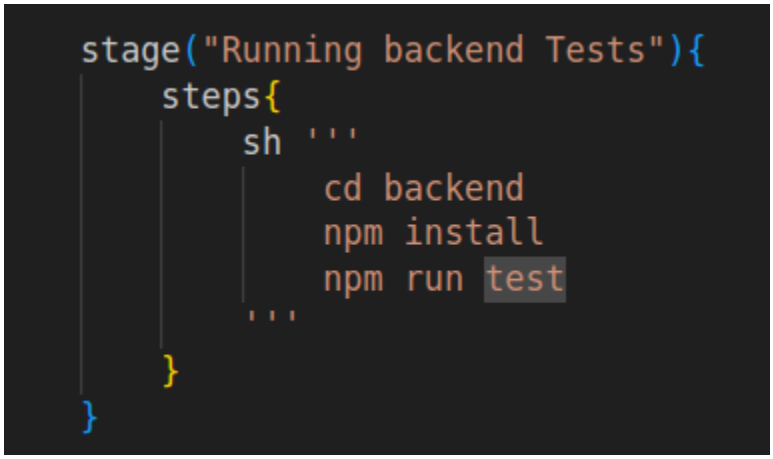
Here, we are cloning the master branch of the Github repository where the project's latest version of codebase exists.

Stage-2:



In this stage we used a multiline shell script in which we cd to the frontend directory and installed all required packages then we ran the front-end testing using React Testing Library.

Stage-3:



We installed all necessary packages then we ran backend testing using the jest library.

Stage-4:

```
stage("Build backend docker image"){
    steps{
        script {
            backend_img = docker.build('chaithanya970/codeeditor:backend', './backend/')
        }
    }
}
```

Here, we are making use of the “Docker” plugin for jenkins to build the back-end image present in the “./backend” directory.

Stage-5:

```
stage("Build frontend docker image"){
    steps{
        script {
            frontend_img = docker.build('chaithanya970/codeeditor:frontend', './frontend/')
        }
    }
}
```

Here, we are making use of the “Docker” plugin for jenkins to build the front-end image present in the “./frontend” directory.

Stage-6:

```
stage("Build coderunner docker image"){
    steps{
        script {
            coderunner_img = docker.build('chaithanya970/codeeditor:coderunner', './coderunner/')
        }
    }
}
```

Here, we are making use of the “Docker” plugin for jenkins to build the coderunner image present in the “./coderunner” directory.

Stage-7:

```
stage("Pushing docker images"){
    steps{
        script {
            docker.withRegistry('', 'Dockerhub') {
                backend_img.push()
                frontend_img.push()
                coderunner_img.push()
            }
        }
    }
}
```

Here, we are pushing all the built images in the previous stages i.e. front-end, back-end and coderunner onto Dockerhub so that the images are available for anyone to pull and use.

Stage-8:

```
        stage("Ansible Deployment") {
            steps {
                ansiblePlaybook becomeUser: null,
                colorized: true,
                credentialsId: 'localhost_cred',
                disableHostKeyChecking: true,
                inventory: 'deployment/inventory',
                playbook: 'deployment/deployment.yaml',
                sudoUser: null
            }
        }
    }
}
```

Here, we have written the settings using which ansible shall deploy onto the managed hosts.

ELK Stack

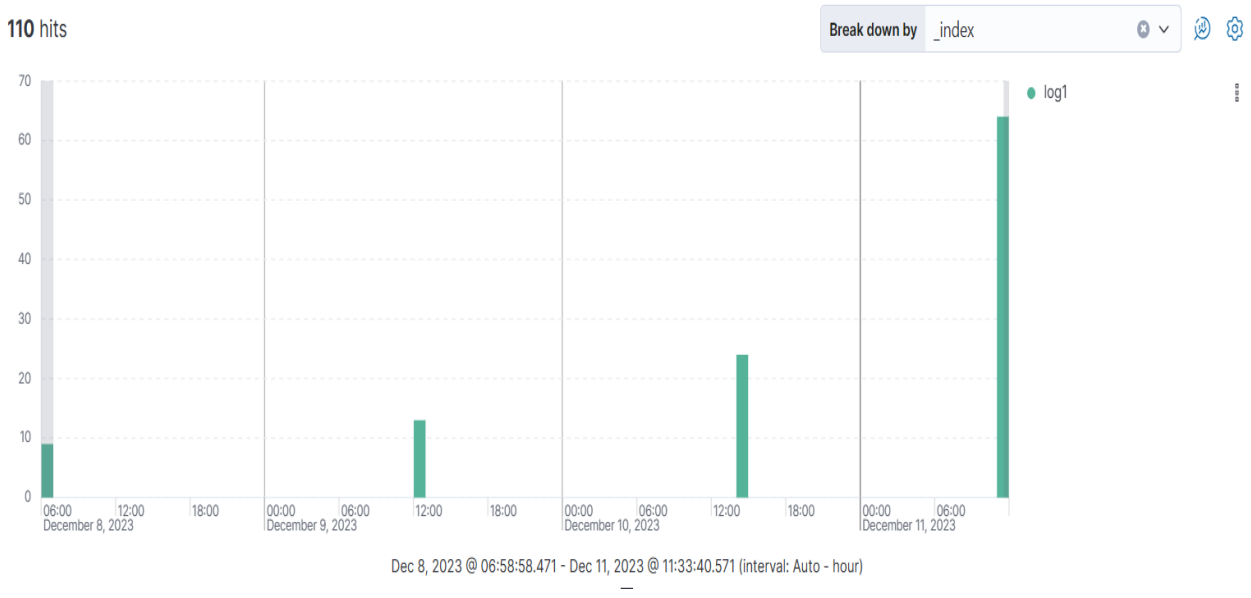
"ELK" is the acronym for three projects: Elasticsearch, Logstash, and Kibana. ELK stack gives us the ability to aggregate logs from all the systems and applications, analyze these logs, and create visualizations.

To use the ELK stack we first require the logs that are generated from the application. Because we have used Docker volumes for persisting the logs as specified in the Docker Compose file, we can get the logs at the path ~/Desktop/Code-Editor/access.log.

Grok pattern:

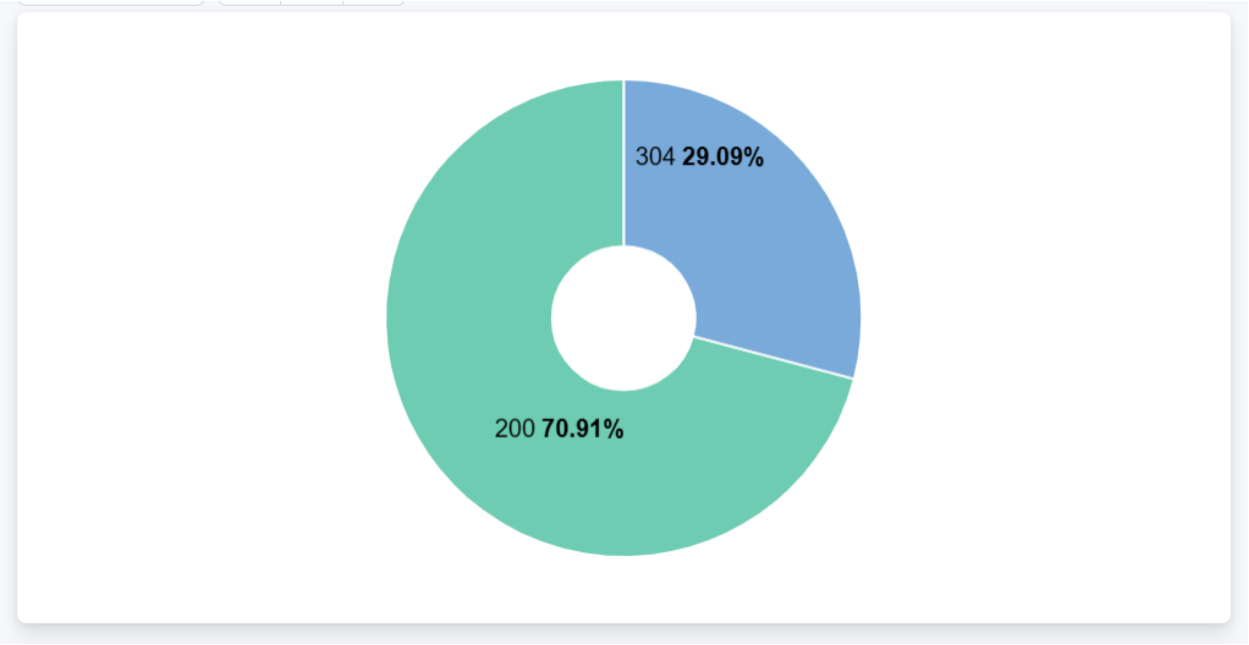
```
%{TIMESTAMP_ISO8601:timestamp} %{WORD:http_request} %{PATH:URLpath}.*?
%{INT:Status_Code} .*? %{NUMBER:Response_time} .*
```

Time Series View

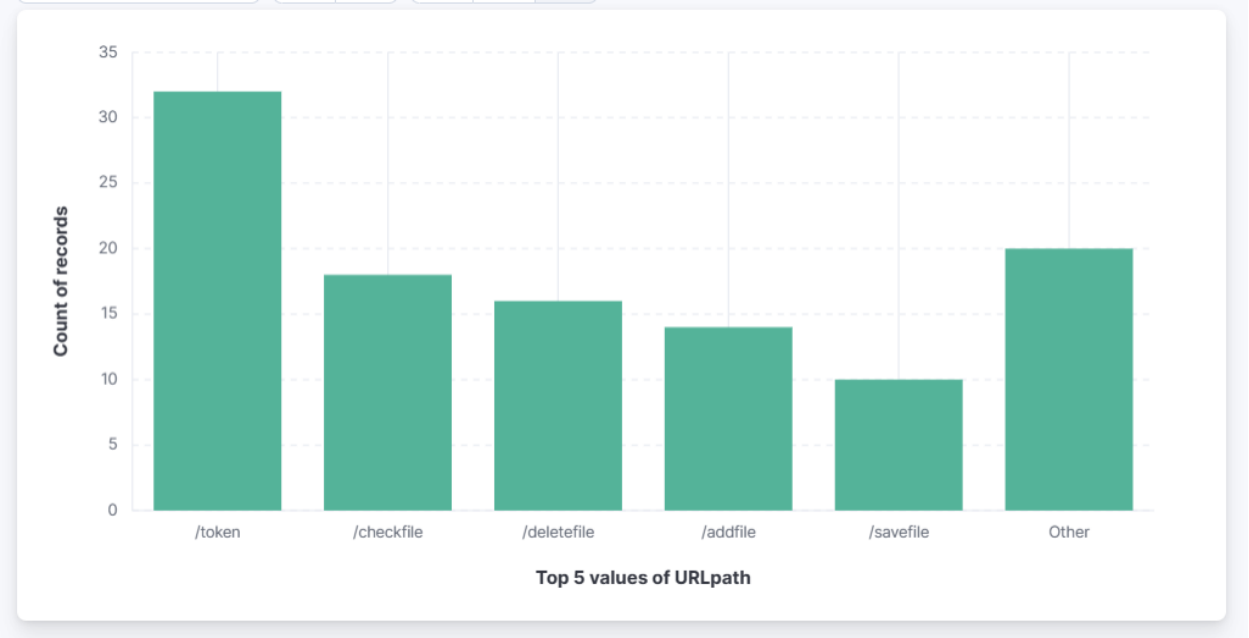


Visualizing http_request field

Visualizing Status_code field



Visualizing URL field



API Documentation

User login API

Endpoint: POST /login

Description:

This API is used for user authentication. It checks the provided email and password with the stored data. If the provided data is valid then it generates the JWT for the user.

Request:

```
body:{
      "Email": "user@gmail.com",
      "Password": "Test@123"
}
```

Responses:

Success response:

```
{
    "Status": "ok",
    "User": "laksdjfljdfkljfkljkldjfsjalksadfisadjfkl"
}
```

Failure response:

```
{
    "Status": "error",
    "Error": "wrong username or password"
}
```

User Register API

Endpoint : POST /createuser

Description:

This api endpoint is used for creating new users. It checks whether provided email exists in the database. If email is not found, then new user is created with the given email, password and name.

Request:

```
{
    "Email": "user@gmail.com",
    "Password": "Test@123",
    "Name": "newuser"
}
```

Responses:

Success Response:

```
{
    "Status" : "ok"
}
```

Failure Response:

```
{
    "Status":"error",
    "Error":"user already exists"
}
```

Token API

Endpoint : GET /token

Description:

This api endpoint is used to verify the given jwt in the header and token is valid then it sends the user information including name, files associated with the user.

Request:

```
{
    "Headers":{
        "X-access-token": token,
    }
}
```

Responses:

Success Response:

```
{
    "Status" :"ok",
    "Files": ["file1","file2"],
    "Username": "user1"
}
```

Failure Response:

```
{
    "Status": "error"
}
```

Add file API

Endpoint: POST /addfile

Description:

This endpoint used to add new file to user files, first it authenticates user using JWT. If user is successfully authenticated then it checks if a file exists in the user files with the given file name, if not then it creates a new file and adds it to the user files.

Request:

```
{
    "Header":{
        "X-access-token":token,
    },
    "Body":{
        "Filename": "file1",
    }
}
```

Response:

Success Response:

```
{
    "Status": "ok",
    "Files": ["file1","file2"]
}
```

Failure Response:

```
{
    "Status": "error",
    "Error": "file already exists"
}
```

Delete file API:

Endpoint : DELETE /deletefile

Description:

This endpoint is used to delete a file. First it authenticates the user using JWT, if it is successful then it removes the file from the user files using the filename.

Request:

```
{
    "Header":{
        "X-access-token":token,
    },
    "Body":{
        "Filename": "file1",
    }
}
```

Response:

Success Response:

```
{
    "Status": "ok",
    "Files": ["file2"]
}
```

Failure Response:

```
{
    "Status": "error",
    "Error": error message
}
```

Getcode API:

endpoint : GET /getcode

Description:

This API endpoint is used to retrieve the unique identifier associated with the specific file from the user files. It verifies the JWT and then it sends the code based on the provided filename.

Request:

```
{
    "Headers":{
        "X-access-token": token,
    },
    "Params":{
        "Filename": "file1",
    }
}
```

Responses:

Success Response:

```
{
    "Status": "ok",
    "Id": id,
}
```

Failure Response:

```
{
    "Status": "error"
}
```

Checkfile API:

Endpoint: GET /checkfile

Description:

This endpoint is used to get the content of a specific file from the user files. It verifies user using JWT, then retrieves file content using provided file identifier.

Request:

```
{
    "Headers":{
        "X-access-token": token,
    },
    "Params":{
        "fileId": id,
    }
}
```

Response:

Success Response:

```
{
    "Status": "ok",
    "Code_content": "cout<<hello;",
    "Username": "user1"
}
```

Failure Response:

```
{
    "Status": "error",
    "Error": "file not found"
}
```

Savefile API:

Endpoint: POST /savefile

Description:

This api endpoint is used to save the file content. It verifies the user using JWT then it retrieves the file using file identifier then it updates the code content of the file then it saves in the database.

Request:

```
{
    {
        "fileId": fileid,
        "Code": content,
    },
    "Headers": {
        "X-access-token":token,
    },
}
```

Response:

Success Response:

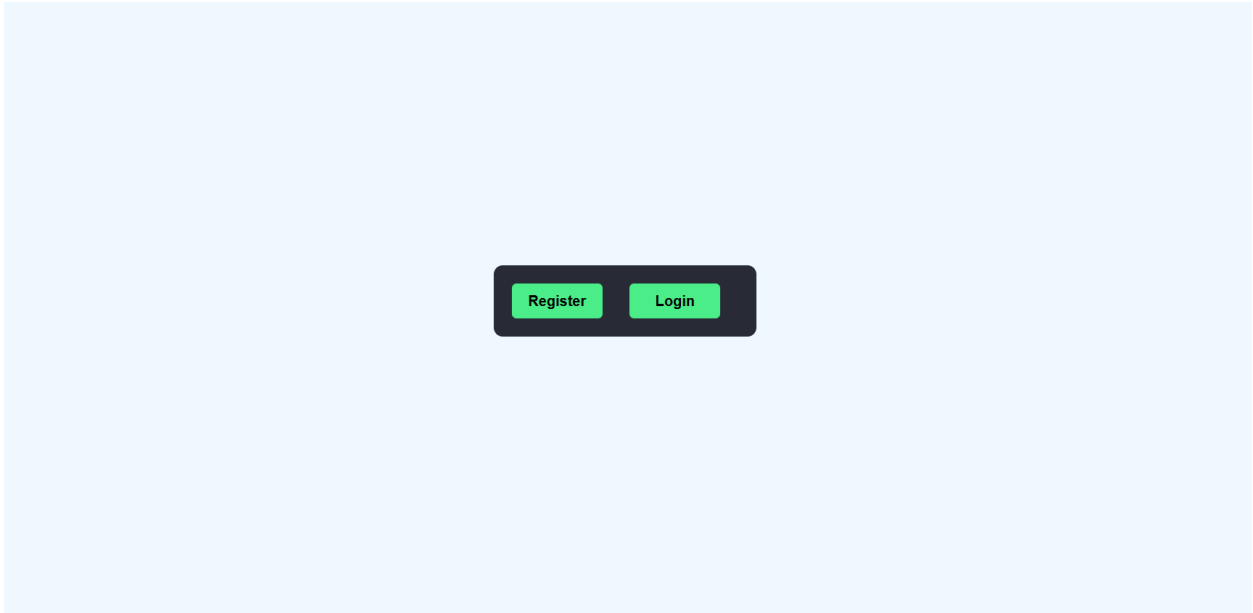
```
{
    "Status": "ok"
}
```

Failure Response:

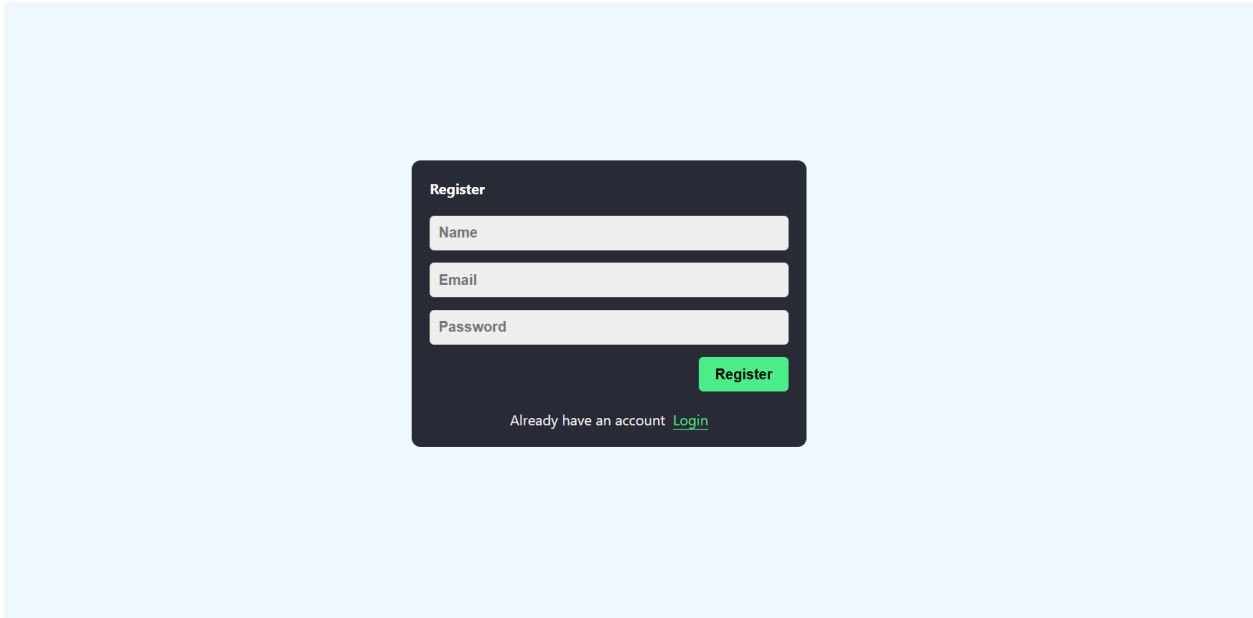
```
{
    "Status": "error"
}
```

Results

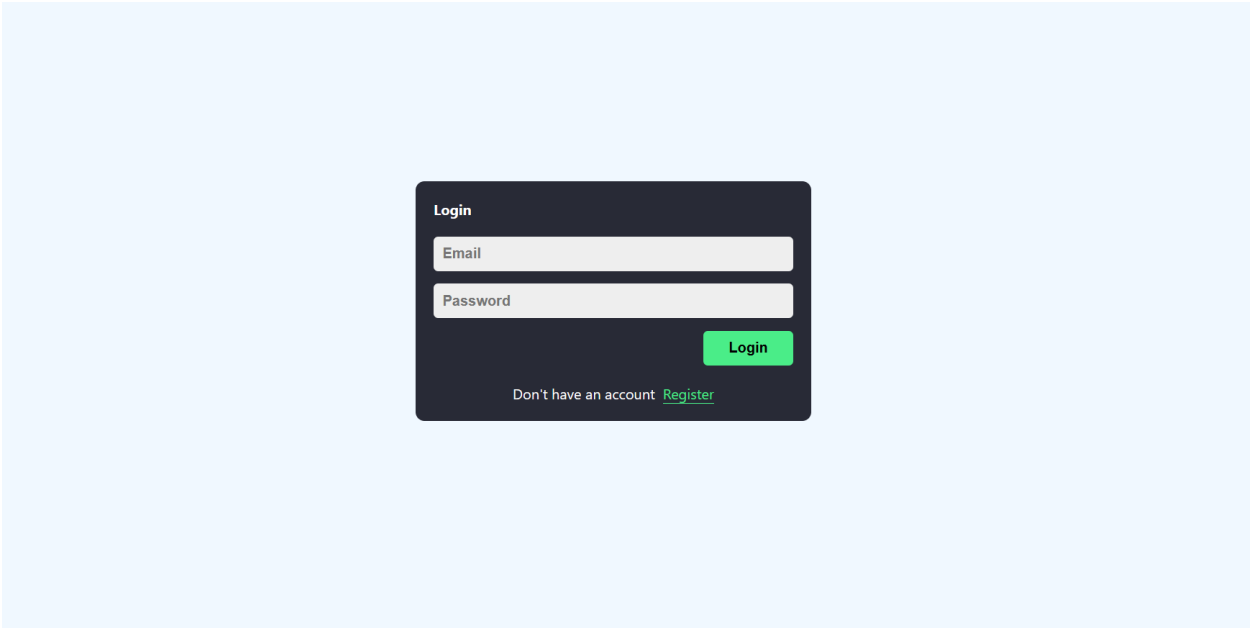
Homepage



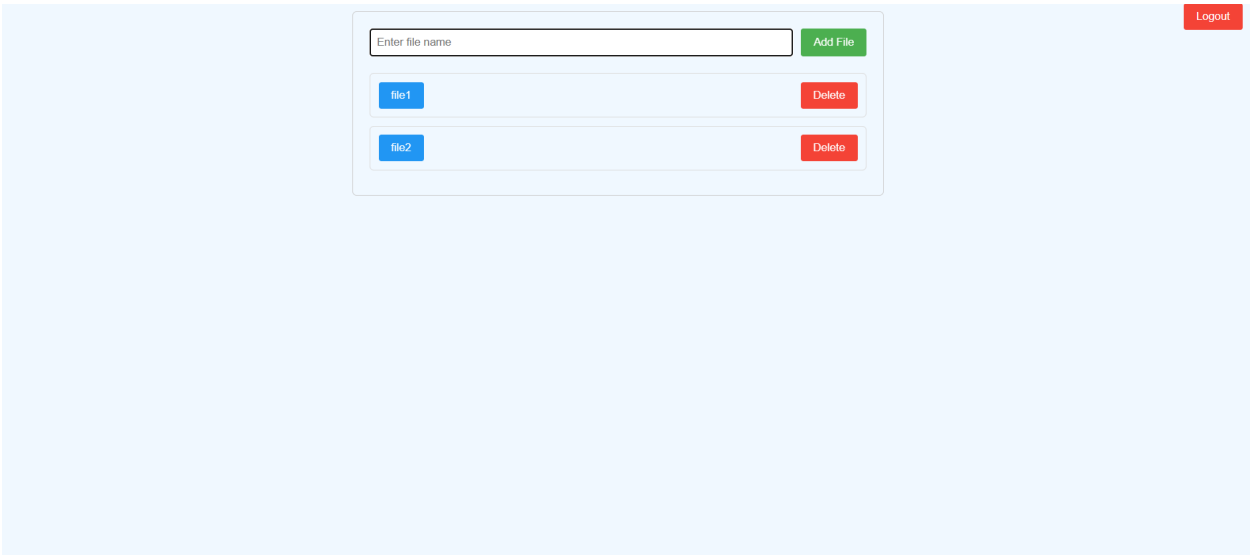
Register Page



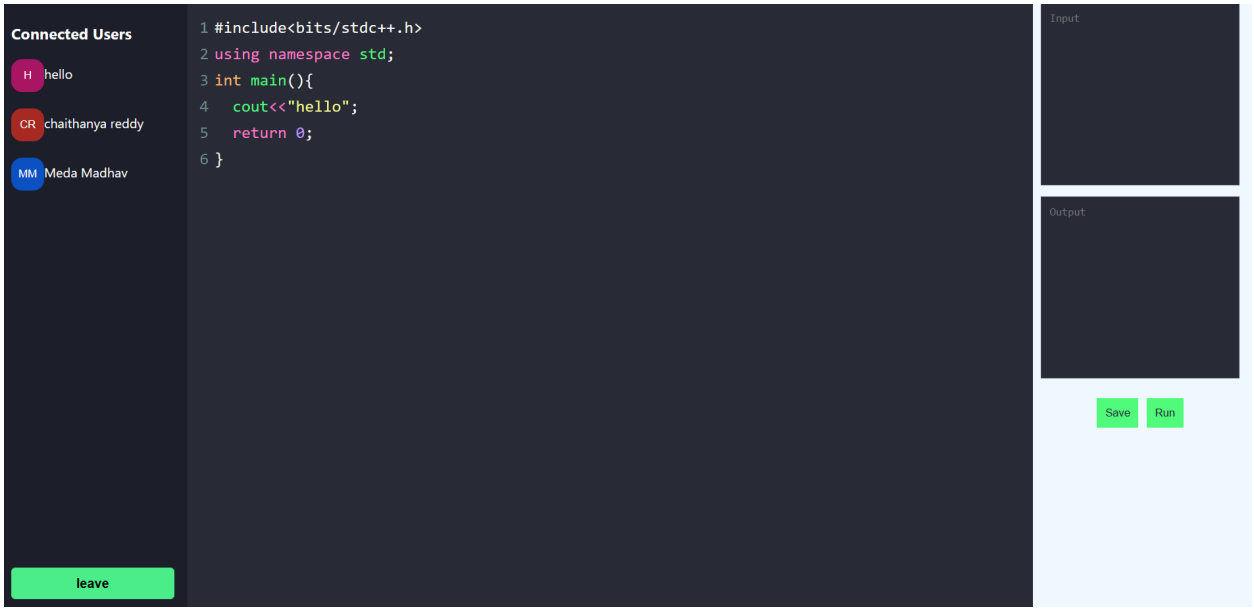
Login Page



Dashboard



Code Editor


Connected Users
H hello
CR chaithanya reddy
MM Meda Madhav
leave
1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4 cout<<"hello";
5 return 0;
6 }
Input
Output
Save
Run