SOFTWARE PRODUCT ENGINEERING

FRIENDZONE

Rachit Agrawal (IMT2020018)

Overview

FriendZone is a social media application designed to bring people closer and make socializing a seamless and enriching experience. In a world where connections matter more than ever, FriendZone offers a platform for users to connect with friends, share moments, and stay updated on each other's lives.

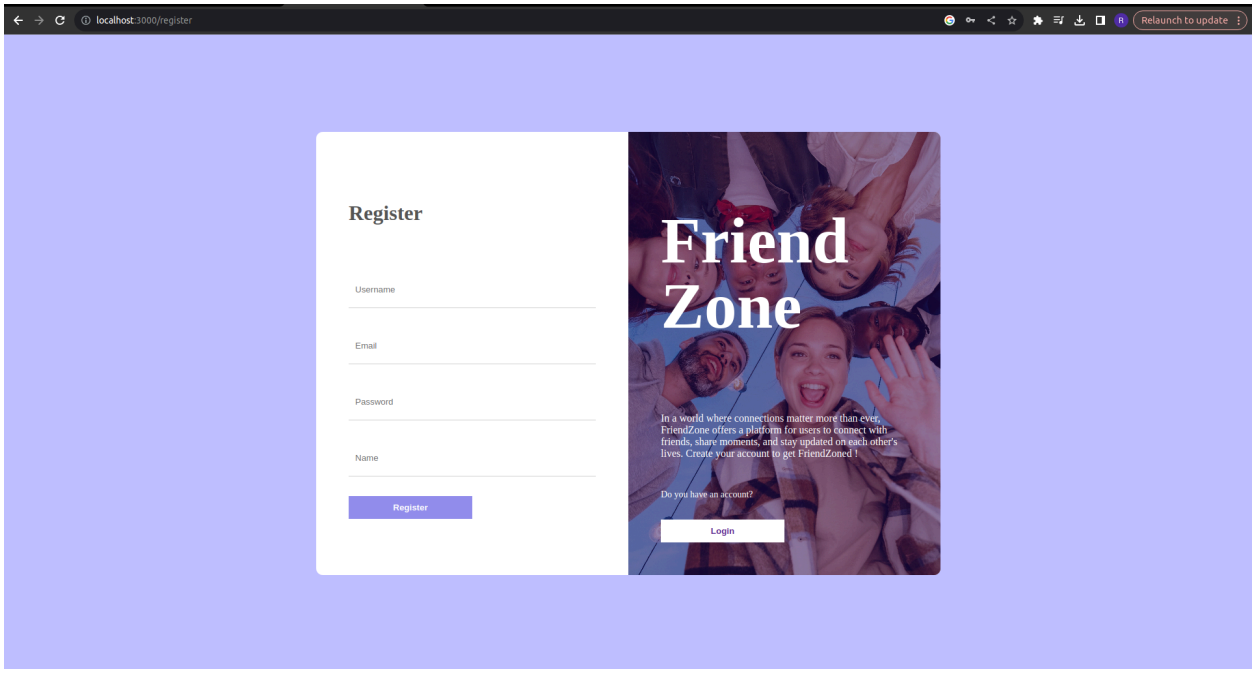
FriendZone provides various features to the user -

- **Secure Registration and Login:** Users can create accounts securely, providing necessary information for registration.
- **Password Security:** Passwords are encrypted using state-of-the-art encryption algorithms, ensuring the utmost security for user accounts.
- **Profile Management:** Users can personalize their profiles by adding a profile picture, cover picture and updating other details, such as name, location, email and website. Easily edit and update profile information as life evolves.
- **User Search and Follow:** Search functionality allows users to find friends or other users based on their usernames. Users can follow each other, building a network of connections and expanding their social circle.
- **Post Creation and Viewing:** Users can create and share posts, including text and photos, to update their friends on their activities and experiences. A dynamic feed allows users to view posts from the people they follow, ensuring they stay connected and informed.
- **Interaction with Posts:** Like and comment on posts to express appreciation or engage in discussions.

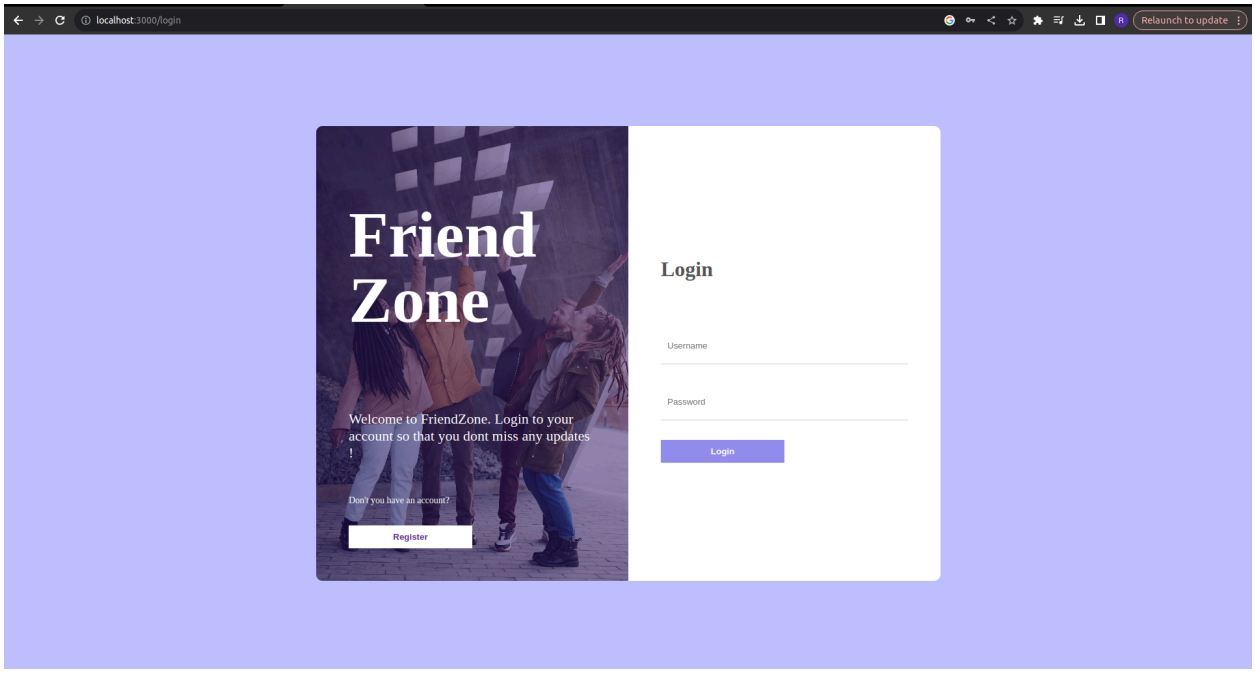
Components

Frontend

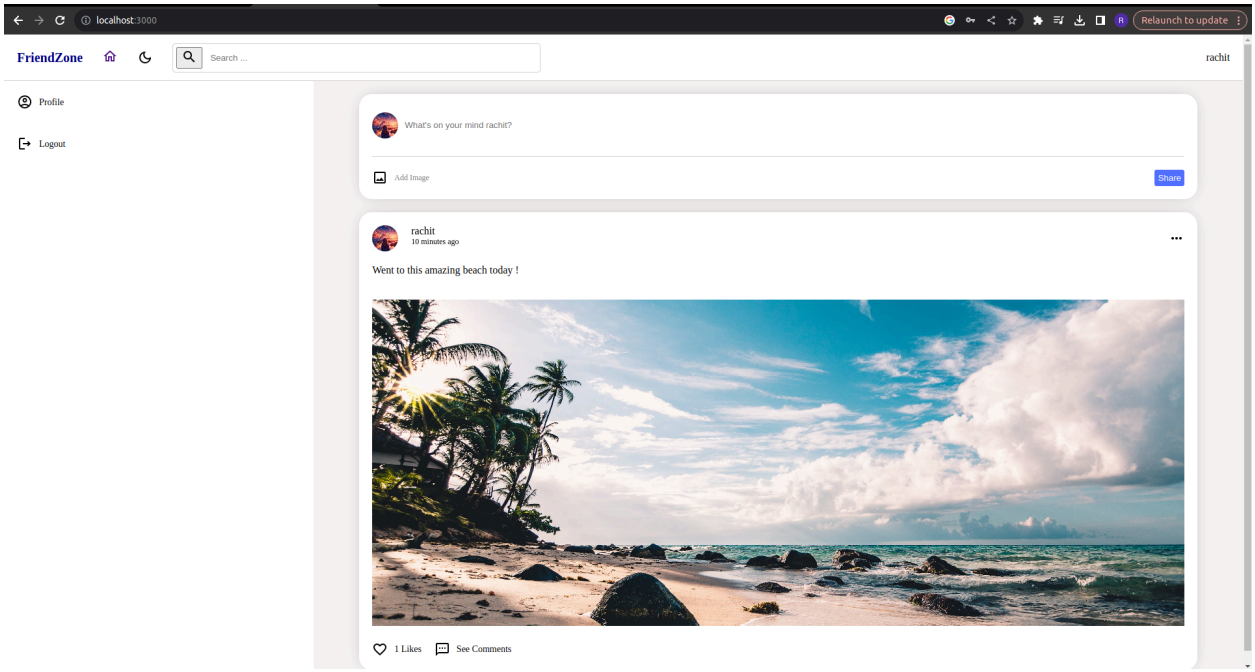
Register Page



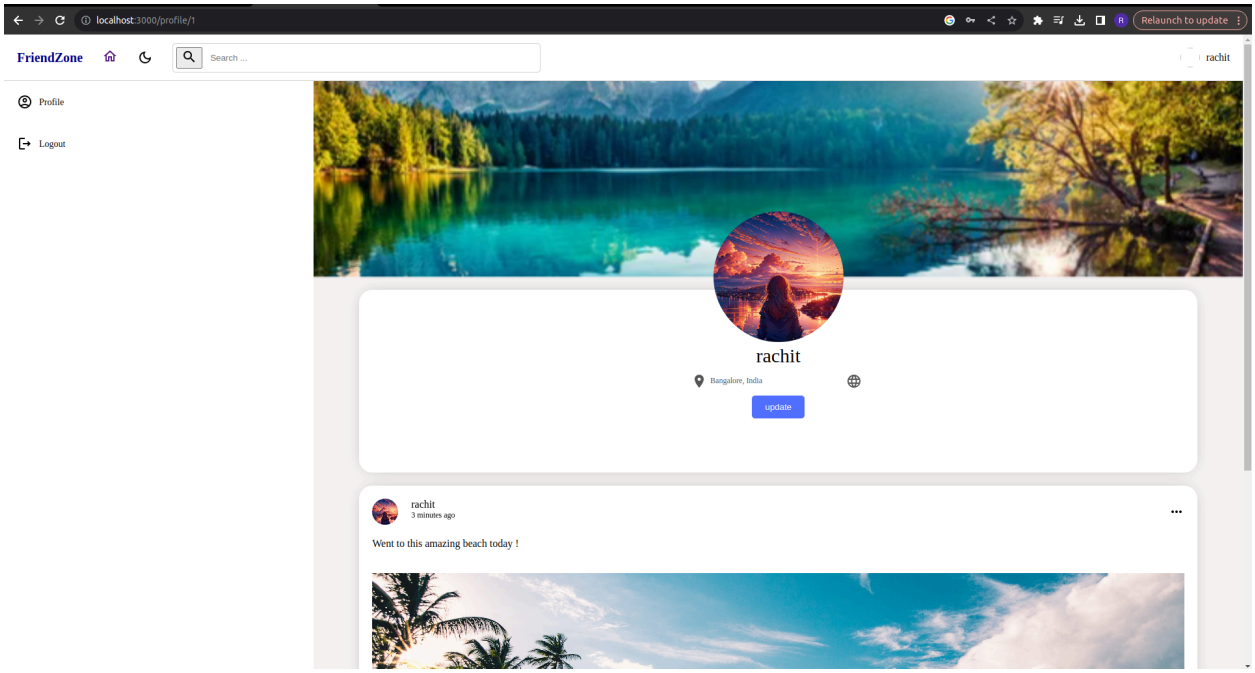
Login Page



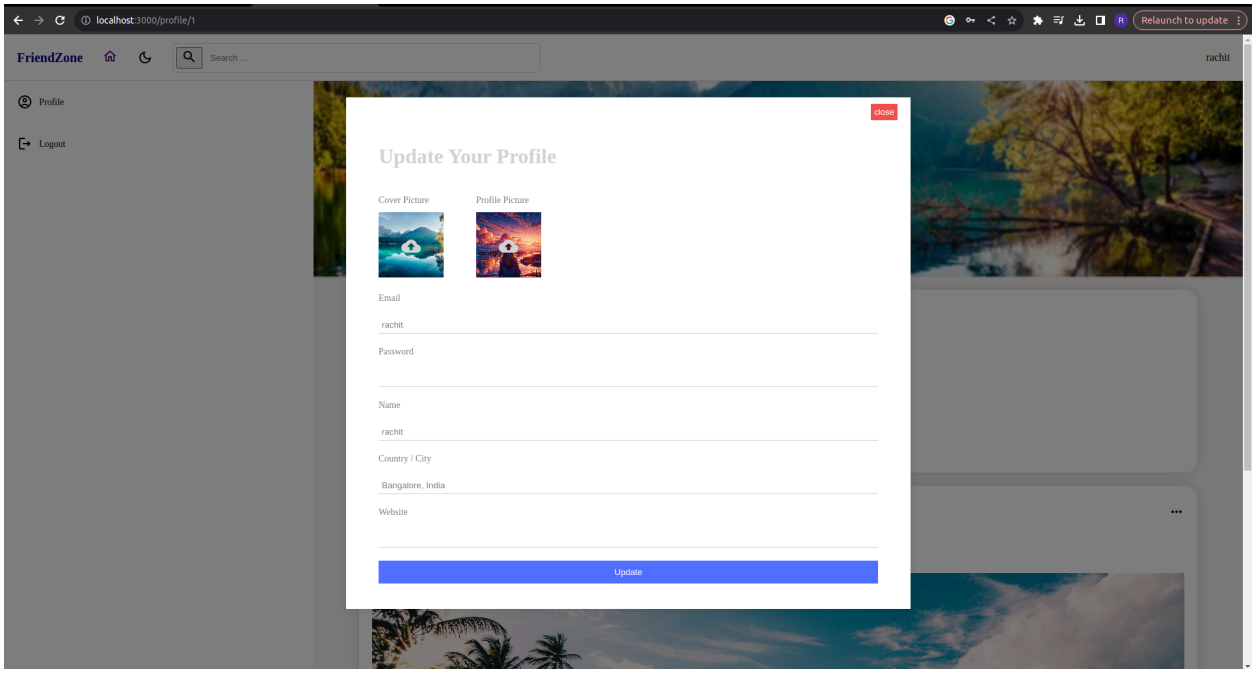
Dashboard



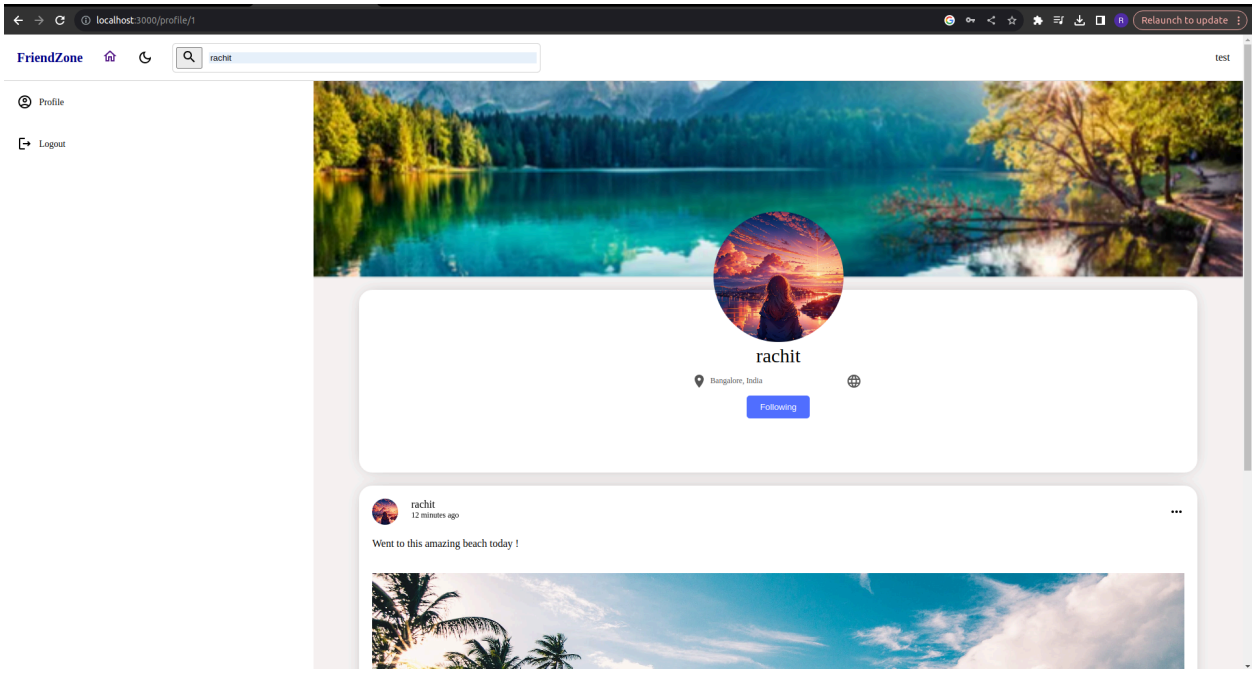
Profile Page



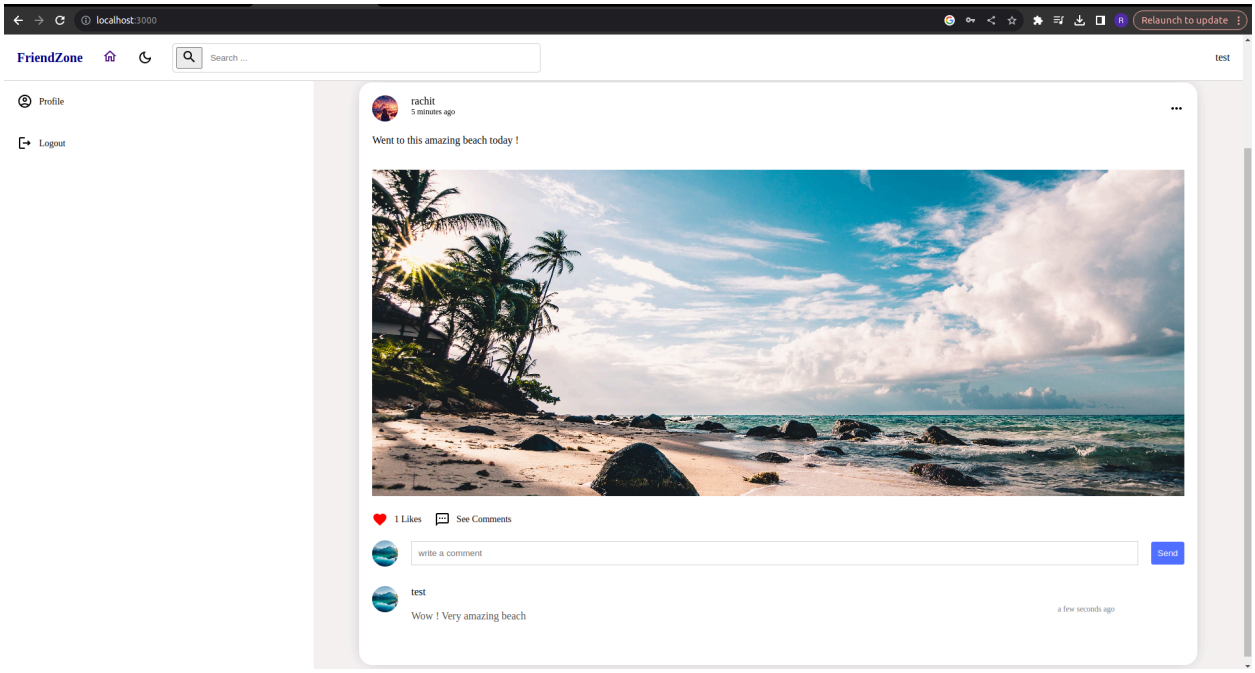
Update Profile



Search Profile



Posts and Comments Section



Backend

MySQL

MySQL is an open-source relational database management system that uses SQL (Structured Query Language) to manage and manipulate data. It is widely used in web applications as a backend database and can handle large amounts of data efficiently. MySQL is known for its speed, reliability, and ease of use, making it a popular choice for developers and businesses alike.

FriendZone uses MySQL for storing various data. The tables used are as follows -

```
+------------------+
| Tables_in_social |
+------------------+
| comments         |
| likes            |
| posts            |
| relationships    |
| users            |
+------------------+
```

Users table

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int          | NO   | PRI | NULL    | auto_increment |
| username   | varchar(45)  | NO   |     | NULL    |                |
| email      | varchar(45)  | NO   |     | NULL    |                |
| password   | varchar(200) | NO   |     | NULL    |                |
| name       | varchar(45)  | NO   |     | NULL    |                |
| coverPic   | varchar(300) | YES  |     | NULL    |                |
| profilePic | varchar(300) | YES  |     | NULL    |                |
| city       | varchar(45)  | YES  |     | NULL    |                |
| website    | varchar(45)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

Posts table

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
desc	varchar(200)	YES		NULL	
img	varchar(200)	YES		NULL	
userid	int	NO	MUL	NULL	
createdAt	datetime	YES		NULL	

Comments table

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
desc	varchar(200)	NO		NULL	
createdAt	datetime	YES		NULL	
userId	int	NO	MUL	NULL	
postId	int	NO	MUL	NULL	

Likes table

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
userId	int	NO	MUL	NULL	
postId	int	NO	MUL	NULL	

Relationships table

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
followerUserId	int	NO	MUL	NULL	
followedUserId	int	NO	MUL	NULL	

Dependencies Used

Server Side Dependencies

- **Express:** A fast, unopinionated, minimalist web framework for Node.js. Used for handling HTTP requests, defining routes, and building the backend server.
- **Bcrypt:** A library for hashing passwords. Commonly used for securely storing user passwords by hashing them before storage.
- **Jsonwebtoken**: Library for creating and verifying JSON Web Tokens (JWT). Enables secure and stateless authentication by generating tokens for user sessions.
- **Cors:** Middleware for handling Cross-Origin Resource Sharing (CORS). Necessary for managing permissions and security when making requests from a frontend app to a backend server on a different domain.
- **cookieParser:** It is a middleware that parses cookies attached to the client's request. Cookies are often used for maintaining session information, user authentication, and other state-related data. cookieParser simplifies the process of parsing and managing cookies in the incoming requests.
- **Mysql:** Used to interact with MySQL databases. It facilitates connecting to the database, executing queries, and managing data.
- **Winston:** Is a versatile logging library for Node.js that supports multiple transports, enabling developers to log messages to various outputs (console, files, databases) with customizable formats and logging levels, making it well-suited for building scalable and flexible logging solutions in Node.js applications.
- **Multer:** It is a middleware for handling multipart/form-data, which is primarily used for uploading files. It is commonly used to handle file uploads. It parses incoming requests that contain file data, making it easier to process and store uploaded files on the server.

Client Side Dependencies

- **React:** JavaScript library for building user interfaces. Provides a component-based architecture for creating reusable UI elements.
- **React Router:** Declarative routing for React.js applications. Enables navigation between different components in a React application.
- **Axios:** A promise-based HTTP client for the browser and Node.js. Used for making API requests from the React frontend to the Node.js backend.
- **Redux:** State management library for JavaScript applications. Helps manage the application's state in a predictable way, especially in larger and more complex applications.
- **react-query:** It is a powerful React library for declarative data fetching, automatic caching, and state synchronization with a server, streamlining the management of server state in React applications. It simplifies API requests with hooks like useQuery and useMutation, offering features like caching, optimistic updates, and pagination.

API Documentation

Register

Register the user with the server and generate a unique id for them.

Method	URL
POST	api/auth/register

Login

Authenticate the user with the system and obtain the auth_token.

Method	URL
POST	api/auth/login

Logout

Logout the user from the system.

Method	URL
POST	api/auth/logout

Search

Search for a user by entering their username.

Method	URL
POST	api/auth/search

Get User Profile

Fetch a user's profile by their userId.

Method	URL
GET	api/users/find/:userId

Update User Profile

Update a user's profile such as name, password, email, profilePic, coverPic, location and website.

Method	URL
PUT	api/users

Fetch Posts

Fetch posts of the current user and of the users followed by the current user.

Method	URL
GET	api/posts

Add a Post

Create a post.

Method	URL
POST	api/posts

Delete a Post

Delete a post.

Method	URL
DELETE	api/posts/:id

Fetch Comments

Fetch comments for a post.

Method	URL
GET	api/comments

Add a Comment

Post a comment.

Method	URL
POST	api/comments

Delete a Comment

Delete a comment.

Method	URL
DELETE	api/comments/:id

Fetch Likes

Fetch likes for a post.

Method	URL
GET	api/likes

Like a Post

Like a post.

Method	URL
POST	api/likes

Unlike a Post

Unlike a post.

Method	URL
DELETE	api/likes

Fetch Relationships

Fetch relationships (followers and following).

Method	URL
GET	api/relationships

Follow a User

Follow a user.

Method	URL
POST	api/relationships

Unfollow a User

Unfollow a user.

Method	URL
DELETE	api/relationships

Testing

Supertest library is used for testing the APIs. It is a popular testing library for Node.js applications that simplifies HTTP assertions and integration testing. It allows developers to make HTTP requests to their server and assert the responses, making it a valuable tool for testing API endpoints and ensuring proper server behavior.

Tests have been performed to check invalid registering and login of users. Other tests are to check that an unauthenticated user does not perform any operations such as creating posts and comments, updating profile, etc. Tests for other functions were not performed as it required authentication of the user through cookies.

Following are the tests performed -

```
const request = require('supertest');
const app = "http://localhost:8000"

describe('Testing api/auth', () => {
    it('Should not login a unregistered user', async () => {
        const details = { username: 'test1', password: 'password' }
        const response = await request(app).post('/api/auth/login').send(details)
        expect(response.status).toBe(404)
    })
    it('Should not register a user with existing username', async () => {
        const details = { username: 'test', password: 'password', email: 'test' , name: 'test'}
        const response = await request(app).post('/api/auth/register').send(details)
        expect(response.status).toBe(409)
    })
    it('Should not login a registered user with wrong password or username', async () => {
        const details = { username: 'test', password: 'passwords' }
        const response = await request(app).post('/api/auth/login').send(details)
        expect(response.status).toBe(400)
    })
    it('Should login a registered user', async () => {
        const details = { username: 'test', password: 'password' }
        const response = await request(app).post('/api/auth/login').send(details)
        expect(response.status).toBe(200)
    })
    it('Should logout', async () => {
        const details = { username: 'test', password: 'password' }
        const response = await request(app).post('/api/auth/logout').send(details)
        expect(response.status).toBe(200)
    })
})
```

```
describe('Testing api/users', () => {
    it('Should not update a user who is not logged in', async () => {
        const details = { name: "test", city: "bangalore", website: "", profilePic: "", coverPic: "" }
        const response = await request(app).put('/api/users').send(details)
        expect(response.status).toBe(401)
    })
})

describe('Testing api/relationships', () => {
    it('Should not follow a user by a user who is not logged in', async () => {
        const details = { userId: "1" }
        const response = await request(app).post('/api/relationships').send(details)
        expect(response.status).toBe(401)
    })
    it('Should not unfollow a user by a user who is not logged in', async () => {
        const details = { userId: "1" }
        const response = await request(app).delete('/api/relationships').send(details)
        expect(response.status).toBe(401)
    })
})
```

```
describe('Testing api/posts', () => {
    it('Should not fetch posts by a user who is not logged in', async () => {
        const details = { userId: "1" }
        const response = await request(app).get('/api/posts').send(details)
        expect(response.status).toBe(401)
    })
    it('Should not create post by a user who is not logged in', async () => {
        const details = { desc: "" , img: ""}
        const response = await request(app).post('/api/posts').send(details)
        expect(response.status).toBe(401)
    })
    it('Should not delete post by a user who is not logged in', async () => {
        const details = { userId: "1" }
        const response = await request(app).delete('/api/posts/1').send(details)
        expect(response.status).toBe(401)
    })
})

describe('Testing api/comments', () => {
    it('Should not post comment by a user who is not logged in', async () => {
        const details = { postId: "1" , desc: ""}
        const response = await request(app).post('/api/comments').send(details)
        expect(response.status).toBe(401)
    })
    it('Should not delete comment by a user who is not logged in', async () => {
        const details = { commentId: "1" }
        const response = await request(app).delete('/api/comments/1').send(details)
        expect(response.status).toBe(401)
    })
})
```

```
describe('Testing api/likes', () => {
    it('Should not like by a user who is not logged in', async () => {
        const details = { postId: "1"}
        const response = await request(app).post('/api/likes').send(details)
        expect(response.status).toBe(401)
    })
    it('Should not delete like by a user who is not logged in', async () => {
        const details = { postId: "1"}
        const response = await request(app).delete('/api/likes').send(details)
        expect(response.status).toBe(401)
    })
})
```

CI/CD and Continuous Monitoring

GitHub Repository - https://github.com/rachit3006/FriendZone

DockerHub - https://hub.docker.com/repositories/rachit3006

Tools Used

- **Git:** a source code management tool that allows developers to collaborate on code and track changes.
- **node.js:** A JavaScript runtime built on Chrome's V8 engine, enabling server-side execution of JavaScript code and facilitating the development of scalable and high-performance web applications.
- **React.js:** A JavaScript library developed by Facebook for building user interfaces. It allows developers to create interactive and reusable UI components.
- **Vite:** Vite is a build tool and development server designed to streamline the development workflow for modern web applications.
- **npm:** The default package manager for Node.js, facilitating the installation, sharing, and management of JavaScript libraries and tools.
- **Jenkins:** a continuous integration and continuous delivery (CI/CD) tool that automates the build, test, and deployment processes.
- **GitHub Webhooks:** a tool that trigger automated actions when specific events occur in a GitHub repository.
- **Docker:** a containerization platform that enables developers to package applications and dependencies into portable, lightweight containers.
- **Docker Compose:** Tool for defining and running multi-container Docker applications, simplifying the process of managing complex, interconnected services within a unified configuration file.
- **Ansible:** a configuration management tool that automates the deployment and management of infrastructure and applications.
- **ELK:** an acronym for Elasticsearch, Logstash, and Kibana, which are open source tools used for centralized logging and log analysis.

- **ngrok:** a tool that creates secure tunnels to expose local servers to the public internet, useful for testing and development purposes.

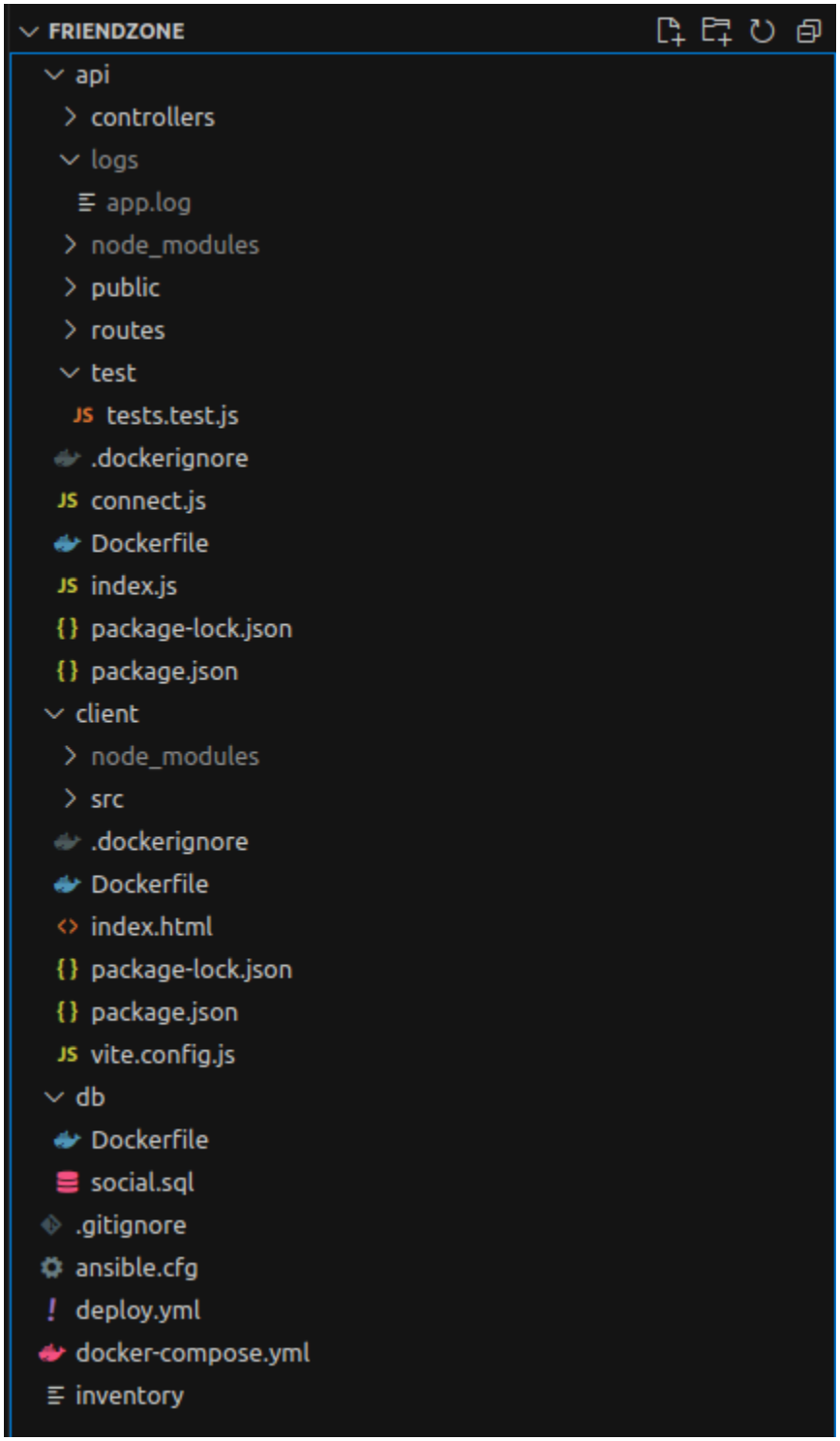
Development and Deployment Process

Code, Build and Test

Tools Used

- Programming Language - JavaScript
- Logging - nodeFileLogger
- Testing - supertest
- Build - node.js

Directory Structure



- **api** - Contains the server side code for the application.
    - **controllers:** It holds modules that define the application's logic. Each file represents a different resource (e.g., users, posts) and contains methods handling specific operations.
    - **routes:** It manages the API routes. It contains modules defining the endpoints and associating them with the corresponding controller methods.
    - **public/upload:** It is used to store the images uploaded by the users for various purposes such as profile picture, post images and cover picture.
    - **test:** It contains the test file used for testing the API.
    - **logs:** It contains the generated logs.
    - **index.js:** It is used as an entry point or main configuration file, consolidating and exporting routes, middleware, or other components to enhance organization and maintainability.
    - **connect.js:** It is used to set up a connection with a MySQL database.
    - **Dockerfile:** It is used to containerize the server.
- **client** - Contains the client side code for the application.
    - **src:** Location for storing the source code of a React application. It contains the main components and pages (such as the login page, register page, home page, navbar, etc.), styles, assets, and other files necessary for building the application.
    - **index.html:** Serves as the entry point for the application, providing the root HTML structure where React components are injected, and it is the initial HTML file loaded by web browsers when rendering the React application.
    - **Dockerfile:** It is used to containerize the client.
- **db -** Contains two files -
    - **Dockerfile:** It is used to containerize the database.
    - **social.sql:** It contains sql commands to create an entrypoint script which is run to initialize the database by creating tables when the database container is first run.

Building and Testing

To build the project, install the node modules in the api and client folder by running the following command in both the folders -

$ npm install

Now the modules required are installed and a node_modules folder would be created in both the folders.

Now for starting the server run the following command in the api folder-

$ npm start

And for starting the client run the following command in the client folder-

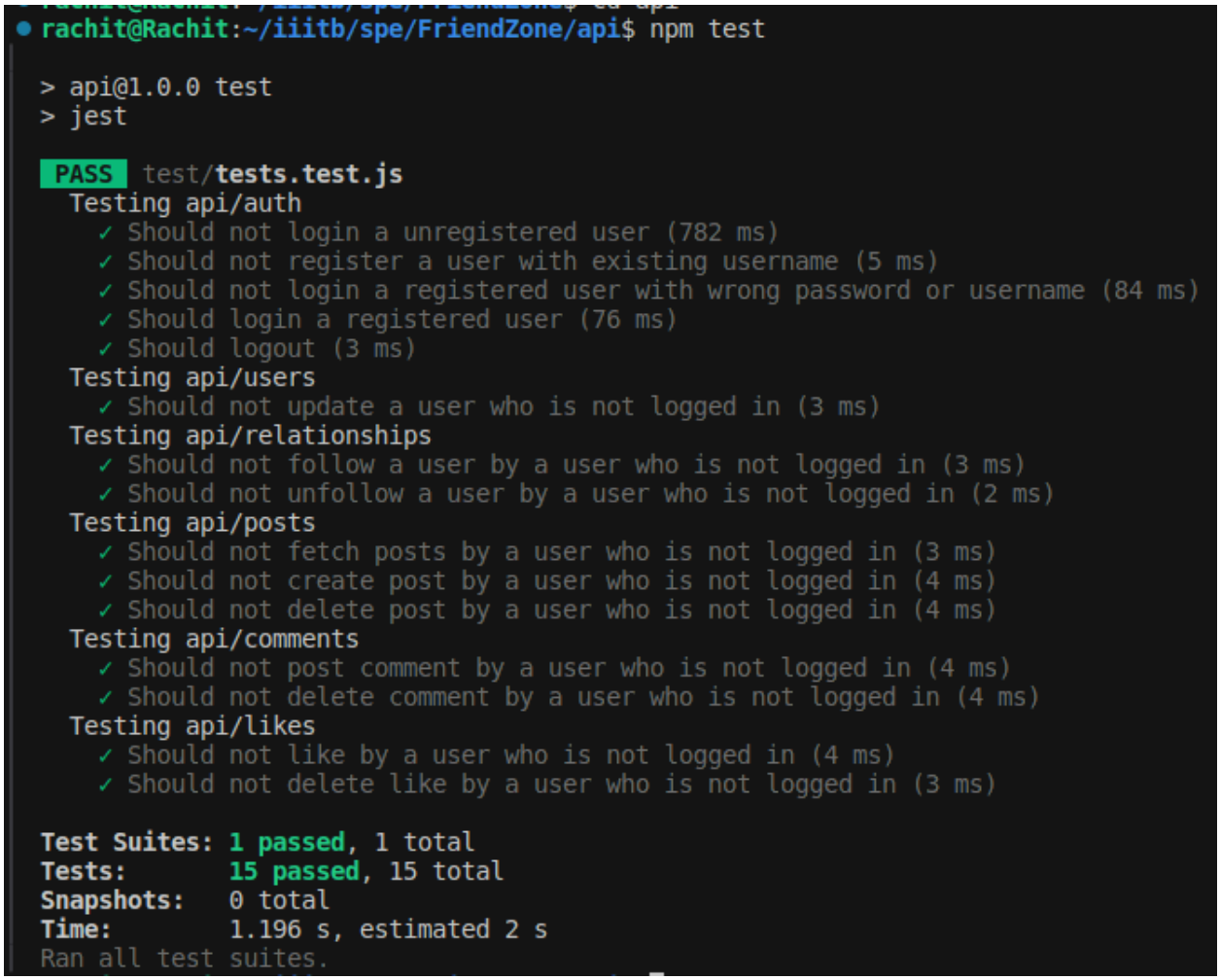
$ npm run dev

For running the tests successfully you need to update the details of the MySQL server in the connect.js file inside the api folder so that the connection to the database is established correctly.

```
import mysql from "mysql"

export const db = mysql.createPool({
  host: process.env.MYSQL_HOST_IP || "localhost",
  user: process.env.MYSQL_USER || "root",
  password: process.env.MYSQL_PASSWORD || "password",
  database: process.env.MYSQL_DATABASE || "social",
});
```

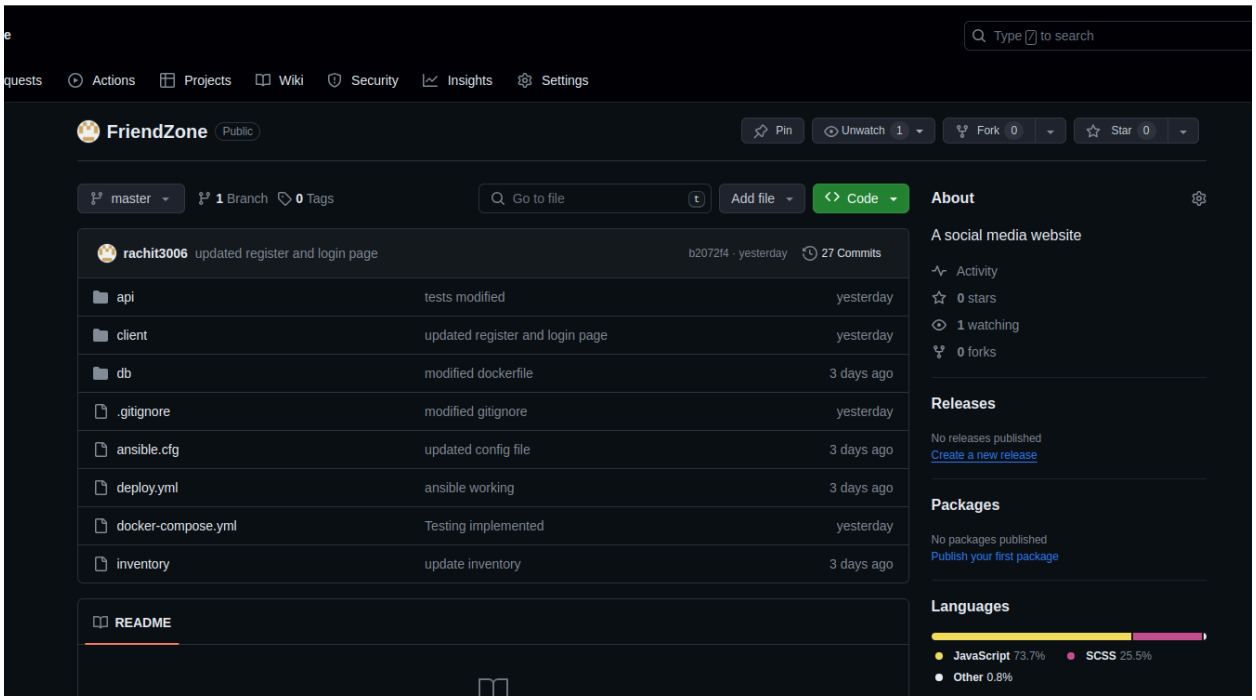
For testing the backend run the following command inside the api folder -

$ npm test


rachit@Rachit:~/iiitb/spe/FriendZone/api$ npm test
> api@1.0.0 test
> jest
PASS test/tests.test.js
Testing api/auth
✓ Should not login a unregistered user (782 ms)
✓ Should not register a user with existing username (5 ms)
✓ Should not login a registered user with wrong password or username (84 ms)
✓ Should login a registered user (76 ms)
✓ Should logout (3 ms)
Testing api/users
✓ Should not update a user who is not logged in (3 ms)
Testing api/relationships
✓ Should not follow a user by a user who is not logged in (3 ms)
✓ Should not unfollow a user by a user who is not logged in (2 ms)
Testing api/posts
✓ Should not fetch posts by a user who is not logged in (3 ms)
✓ Should not create post by a user who is not logged in (4 ms)
✓ Should not delete post by a user who is not logged in (4 ms)
Testing api/comments
✓ Should not post comment by a user who is not logged in (4 ms)
✓ Should not delete comment by a user who is not logged in (4 ms)
Testing api/likes
✓ Should not like by a user who is not logged in (4 ms)
✓ Should not delete like by a user who is not logged in (3 ms)
Test Suites: 1 passed, 1 total
Tests: 15 passed, 15 total
Snapshots: 0 total
Time: 1.196 s, estimated 2 s
Ran all test suites.

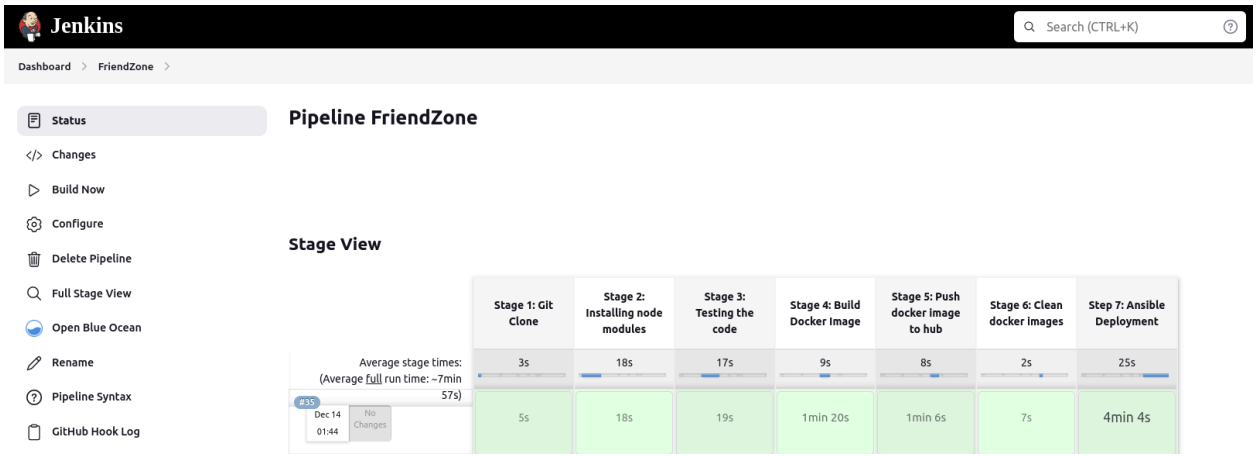
Version Control using GitHub

- Git tracks changes to source code, allowing developers to version their work over time. This enables easy rollbacks to previous states and facilitates collaboration.
- Git logs detailed information about each commit, including the author, timestamp, and a commit message. This makes it easy to understand the history of changes and trace the evolution of the codebase.
- Git enables developers to roll back to previous states of the project easily. This "time travel" feature is invaluable for identifying and rectifying errors or unexpected issues.
- Git integrates seamlessly with Continuous Integration/Continuous Deployment (CI/CD) pipelines. Automated builds, tests, and deployments can be triggered based on code changes, ensuring a streamlined development and release process.



DOCKER-COMPOSE AND JENKINS

- Jenkins is an open-source automation tool written in Java with plugins built for continuous integration.
- Jenkins is utilized to continuously build and test software projects, simplifying the process for developers to integrate changes, and allowing users to obtain up-to-date builds with ease.
- Additionally, Jenkins enables continuous delivery of software by integrating with a wide range of testing and deployment technologies.
- We utilize Jenkins Pipeline for our project. Jenkins Pipeline is a suite of plugins that allows developers to create and manage continuous delivery pipelines as code. It enables developers to define the entire software delivery process in a single pipeline script.



Jenkins Pipeline

```
pipeline{
 environment{
    docker_image = ""
    dpass = "PaPa12345"
    registry1 = "rachit3006/friendzone_mysql"
    registry2 = "rachit3006/friendzone_server"
    registry3 = "rachit3006/friendzone_client"
 }
 agent any
 stages{
    stage('Stage 1: Git Clone'){
        steps{
            git branch: 'master',
            url:'https://github.com/rachit3006/FriendZone'
        }
    }
    stage('Stage 2: Installing node modules'){
        steps{
            script{
                dir('api'){
                    sh "npm install"
                }
                dir('client'){
                    sh "npm install"
                }
            }
        }
    }
    stage('Stage 3: Testing the code'){
        steps{
            script{
                dir('api'){
                    sh "forever start index.js"
                    sh "npm test"
                    sh "forever stop index.js"
                    sh "rm -rf node_modules"
                    sh "sudo rm -rf logs"
                }
                dir('client'){
                    sh "rm -rf node_modules"
                }
            }
        }
    }
    stage('Stage 4: Build Docker Image'){
        steps{
            script{
                docker_image = sh "docker-compose build"

            }
        }
    }
```

```
    stage('Stage 5: Push docker image to hub') {
        steps{
            script{
                sh "docker login -u 'rachit3006' -p " + dpass
                sh "docker push " + registry1 + ":latest"
                sh "docker push " + registry2 + ":latest"
                sh "docker push " + registry3 + ":latest"            }
        }
    }
    stage('Stage 6: Clean docker images') {
        steps{
            script{
                sh 'docker container prune -f'
                sh 'docker image prune -f'

            }
        }
    }
    stage('Step 7: Ansible Deployment'){
        steps{
            sh 'ansible-playbook deploy.yml -i inventory'
        }
    }
}
}
```

1. **Git Clone:** It pulls the remote repository from GitHub using Jenkins.

2. **Installing node modules:** It installs the node modules for the server and the client.

3. **Testing the code:** It tests the server code.

4. **Build Docker Images:** It is used to create images on our local system that are then uploaded to our Docker hub, allowing us to fetch the image and execute the application on other servers. It is built using docker-compose.

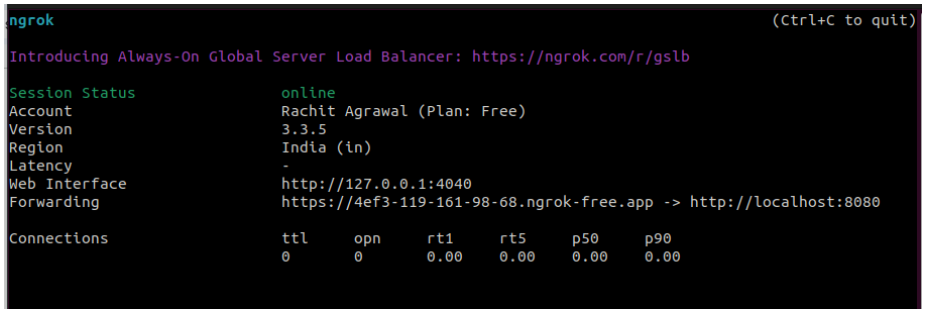
5. **Push Docker Images to Hub:** Deploys the images into DockerHub so that anyone can pull the image.

6. **Clean Docker Images:** The prune command is used to remove the unneeded containers and images.

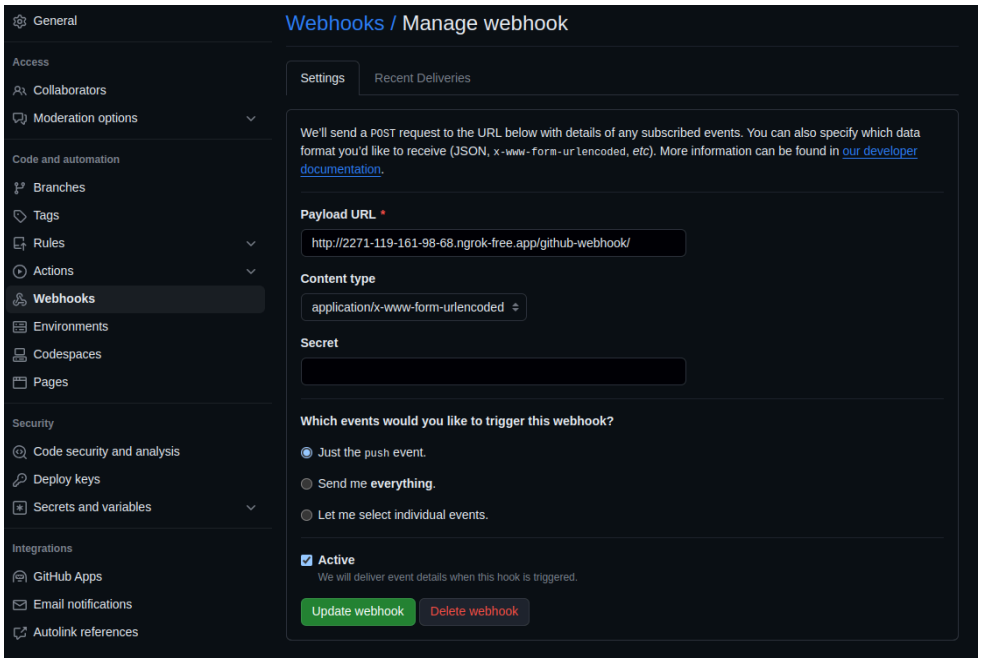
7. **Ansible Deployment:** It runs the Ansible playbook to deploy the application to the hosts specified in the inventory file.

GitHub Webhook

Webhooks are messages that are transmitted immediately whenever there is a change in the environment. If we make any changes to the GitHub repository, the webhook will activate the Jenkins pipeline immediately, and it will do so automatically. Ngrok is able to link local servers that are protected by NATs (Network Address Translation) and firewalls to the public internet by utilizing encrypted tunnels. It is equipped with a real-time web interface that gives you the ability to view any HTTP traffic that is moving through your tunnels. It gives you the capability of connecting to the internet through a web server that is operating on your local machine. Just provide ngrok with the port number on which your web server is actively listening.



In the github repository go to settings and in Webhooks add ngrok address in payload url.



Containerize

- Docker is a platform that simplifies the process of building, deploying, and running applications inside containers. Containers are lightweight, portable, and self-contained environments that provide an isolated environment for running software. Docker allows developers to package their applications and dependencies into a single container that can be deployed across different environments, making it easier to manage and scale applications in a consistent and repeatable manner.
- Docker Compose is a tool that simplifies the management of multi-container Docker applications by defining services, networks, and volumes in a single, declarative configuration file. With a straightforward YAML syntax, Docker Compose enables developers to specify the services required for an application, their configurations, and how they interact. This allows for efficient orchestration of complex architectures, making it easy to deploy, scale, and manage containerized applications with a single command.

Following is the docker-compose.yml file -

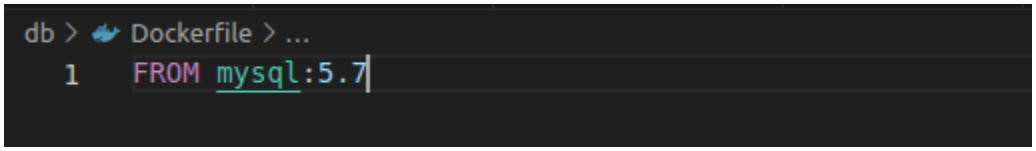
```yaml

version: "3.4"

x-common-variables: &common-variables
  MYSQL_USER: sampleuser
  MYSQL_PASSWORD: samplepassword
  MYSQL_DATABASE: socialdb
  REACT_APP_SERVER_PORT: 8000

services:
  mysql:
    image: "rachit3006/friendzone_mysql:latest"
    build: ./db
    environment:
      <<: *common-variables
      MYSQL_HOST: localhost
      MYSQL_ROOT_PASSWORD: root
    ports:
      - 3307:3306
    restart: unless-stopped
    volumes:
      - ./db/storage:/var/lib/mysql
      - ./db/social.sql:/docker-entrypoint-initdb.d/social.sql
```

```
24    server:
25      image: "rachit3006/friendzone_server:latest"
26      build: ./api
27      depends_on:
28        - mysql
29      expose:
30        - 8000
31      environment:
32        <<: *common-variables
33        MYSQL_HOST_IP: mysql
34      ports:
35        - 8000:8000
36      volumes:
37        - ./api:/app
38      links:
39        - mysql
40      command: npm start
41    client:
42      image: "rachit3006/friendzone_client:latest"
43      build: ./client
44      environment:
45        <<: *common-variables
46        NODE_PATH: src
47      expose:
48        - 3000
49      ports:
50        - 3000:3000
51      volumes:
52        - ./client/src:/app/src
53        - ./api/public/upload:/app/public/upload
54      links:
55        - server
56      command: npm run dev
```

This Docker Compose configuration orchestrates a multi-container setup for the application, composed of three services: MySQL for the database, a Node.js server, and a React.js client. Key configurations include:

MySQL Service:

- Specifies the DockerHub image to be used ("rachit3006/friendzone_mysql:latest").
- Builds from the "./db" directory. Dockerfile used to build -



    1. Sets the base image to mysql:5.7
- Defines common environment variables for MySQL, including user, password, and database name.
- Maps ports (3307:3306) and specifies volume mounts for MySQL storage and initialization script.
- Restarts unless explicitly stopped.

Server Service:

- Specifies the DockerHub image to be used ("rachit3006/friendzone_server:latest").
- Builds from the "./api" directory. Dockerfile used to build -

1. Sets the base image to Node.js with Alpine Linux, a lightweight distribution.
2. Sets the working directory within the container to "/app".
3. Creates a "public/upload" directory inside the "/app" directory for storing the uploaded images.
4. Copies the package.json and package-lock.json files to the "/app" directory.
5. Installs Node.js dependencies based on the copied package files.
6. Copies the entire application code to the "/app" directory.
7. Exposes port 8000, indicating that the application inside the container will listen on this port.
8. Specifies the default command to run when the container starts, initiating the application using "npm start".

- Depends on the MySQL service.
- Exposes port 8000 and maps it to the host machine.
- Sets environment variables for the server, including the MySQL host IP.
- Specifies volume mounts for the server code.

Client Service:

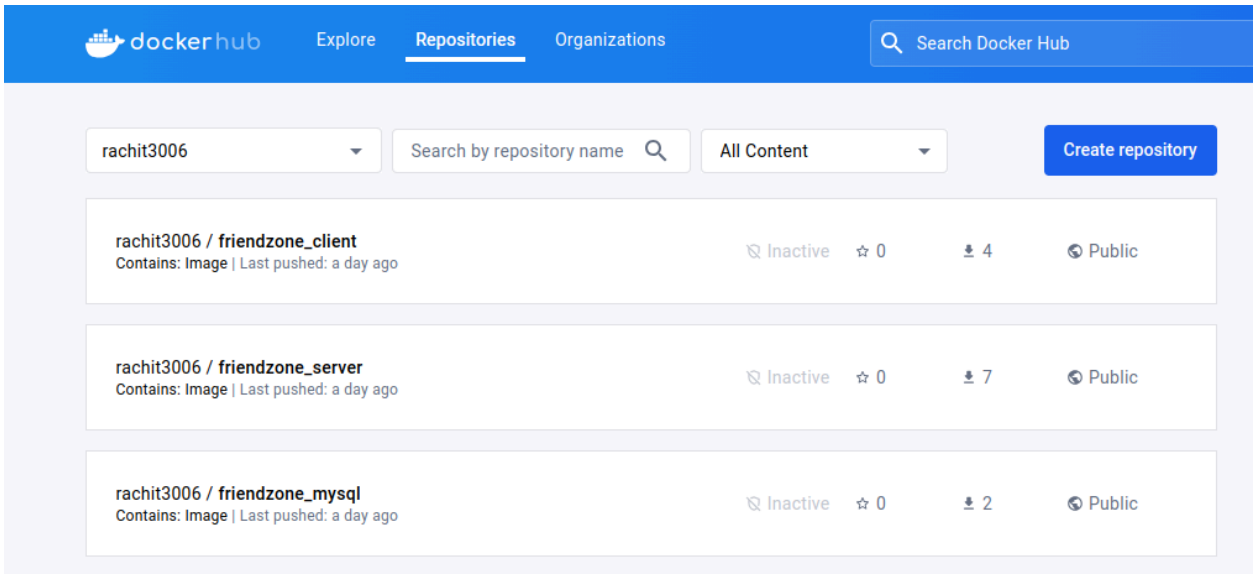
- Specifies the DockerHub image to be used ("rachit3006/friendzone_client:latest").
- Builds from the "./client" directory. Dockerfile used to build -

```
1   FROM node:alpine
2
3   RUN mkdir -p /app
4   WORKDIR /app
5
6   COPY package.json /app
7   COPY package-lock.json /app
8   COPY . /app
9
10  RUN npm install
11
12  CMD ["npm", "run", "dev"]
```

    - Sets the base image to Node.js with Alpine Linux, a lightweight distribution.
    - Sets the working directory within the container to "/app".
    - Copies the package.json and package-lock.json files to the "/app" directory.
    - Installs Node.js dependencies based on the copied package files.

    - Copies the entire application code to the "/app" directory.
    - Specifies the default command to run when the container starts, initiating the application using "npm run dev".
- Exposes port 3000 and maps it to the host machine.
- Sets environment variables for the client, including the Node.js path.
- Specifies volume mounts for the client code and links to the server service.
- Runs the "npm run dev" command.

Following is the DockerHub Repository -



Ansible

Ansible is an open-source automation tool that simplifies IT configuration management and application deployment across servers, networks, and cloud environments using a simple, human readable language. It allows users to automate tasks across multiple systems, making it easier to manage and scale infrastructure. Ansible uses a push-based model, where the control machine pushes configurations and commands to remote machines, and supports a wide range of modules and plugins for managing different technologies and platforms. Ansible is used by IT teams of all sizes to automate repetitive tasks, improve efficiency, and reduce errors in the IT environment.

Playbook

```
! deploy.yml
1   ---
2   - name: Pull Docker Image
3     hosts: all
4     vars:
5       ansible_python_interpreter: /usr/bin/python3
6     tasks:
7       - name: Copy from host machine to remote host
8         copy:
9           src: ./
10          dest: ./app
11      - name: Pull the Docker images specifed in docker-compose
12        shell:
13          cmd: docker-compose pull
14          chdir: ./app
15      - name: Running container
16        shell:
17          cmd: docker-compose -f ./docker-compose.yml up -d
18          chdir: ./app
```

This Ansible playbook automates the process of pulling and deploying a Docker image using Docker Compose. The following steps are executed:

Copy from Host to Remote Host:

Copies the content of the current directory (source: ./) from the Ansible control machine to the remote host's "./app" directory.

Pull Docker Images:

Uses the **docker-compose pull** command to pull Docker images specified in the Docker Compose file. This step ensures that the latest versions of the specified images are available on the remote host. The **chdir** option sets the working directory to "./app" before executing the command.

Run Docker Containers:

Executes the **docker-compose -f ./docker-compose.yml up -d** command to run Docker containers in detached mode. This step launches the containers defined in the Docker Compose file. The **chdir** option sets the working directory to "./app" before executing the command.

Inventory

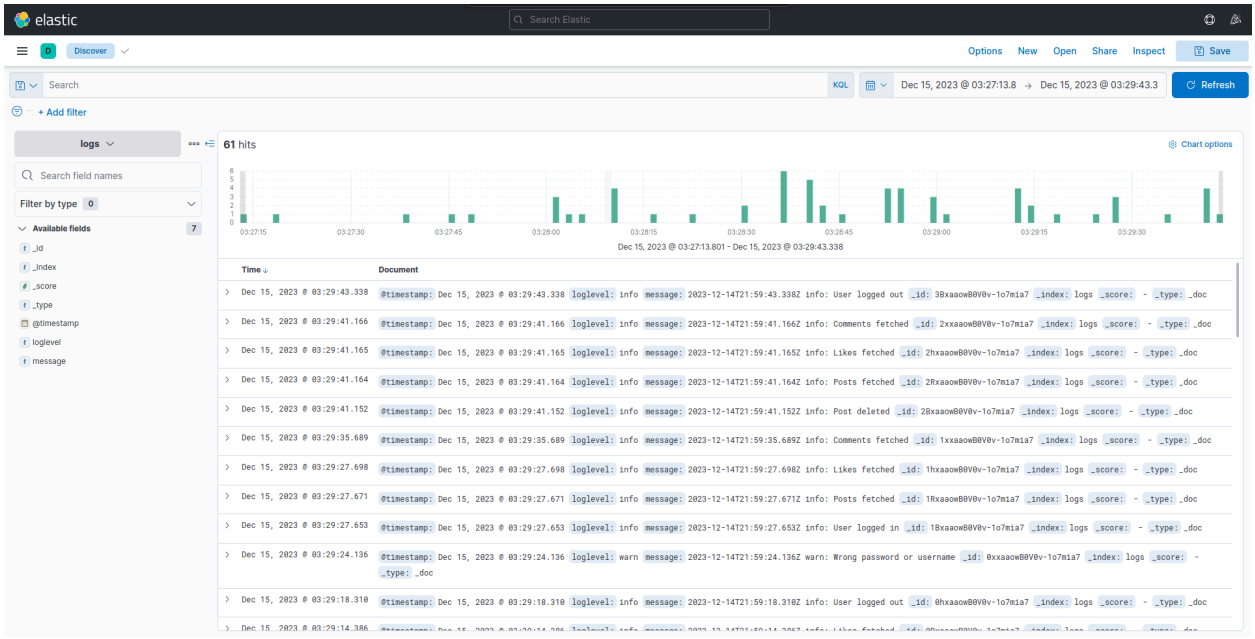
inventory

```
1  [myhost]
2  localuser ansible_host=192.168.80.1 ansible_user=localuser  ansible_ssh_password=ritika123
```

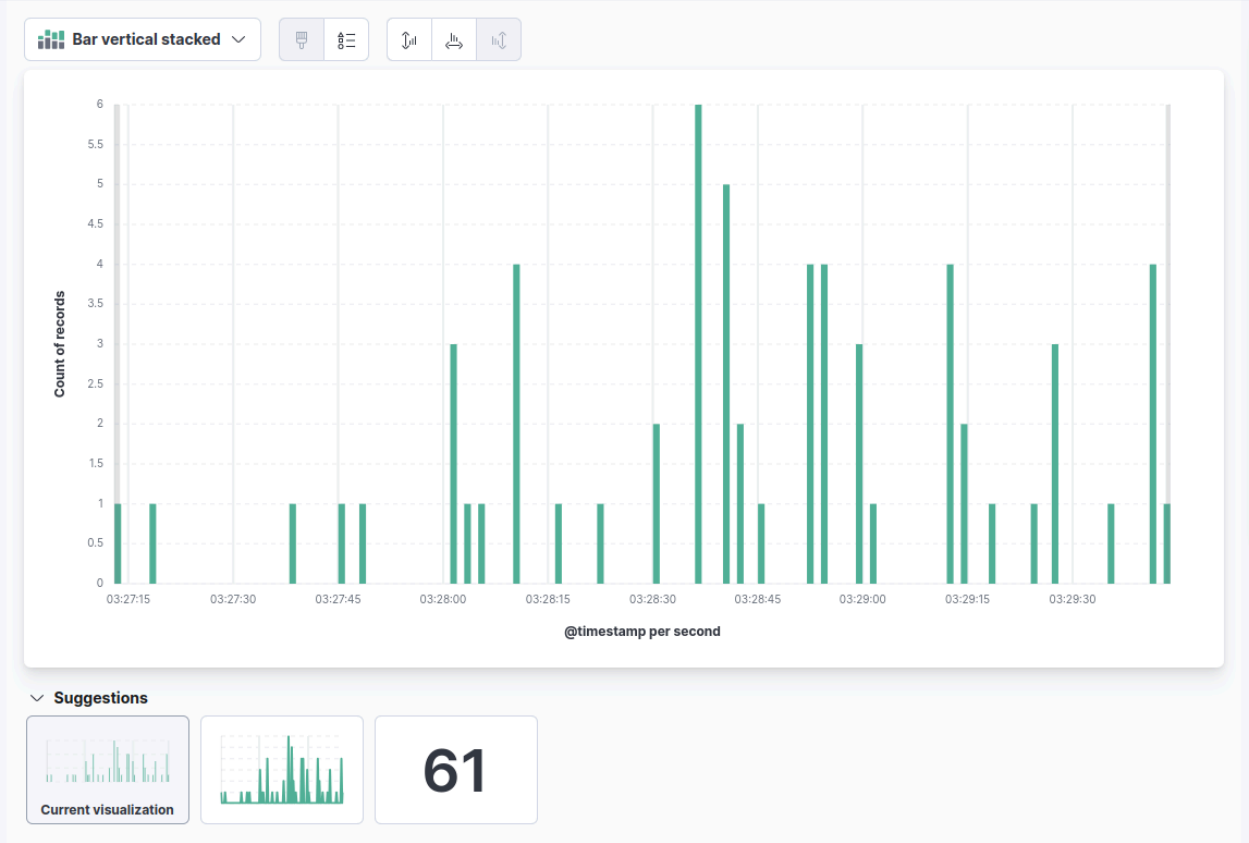
It specifies the host on which the application needs to be deployed. In my case it is “localuser@192.168.80.1” which is another user on my system.

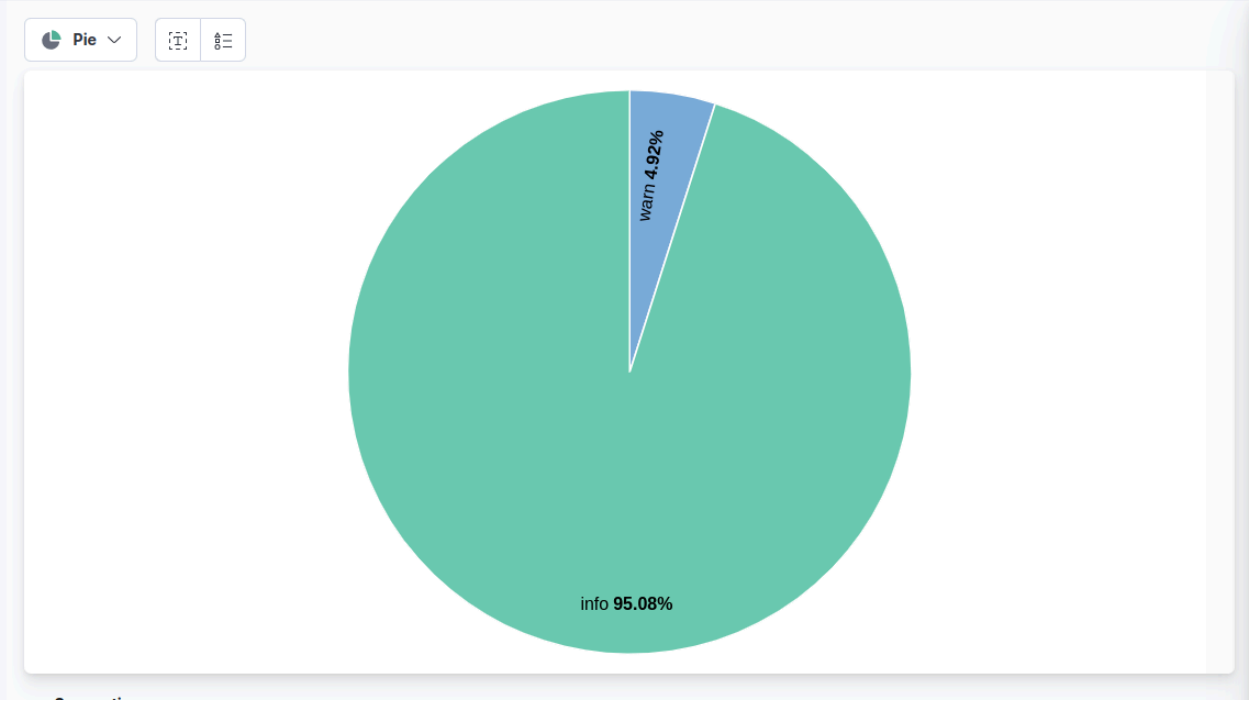
Monitoring

Monitoring involves checking to see if the program is operating as planned. ELK stack is the monitoring tool that has been used to test the deployed application. It does an analysis of the generated logs, and then that analysis can be viewed on the kibana dashboard.



Following are the visualizations generated by kibana using the logs -





Challenges and Errors Faced

- First challenge was connecting the MySQL docker container with the backend container and setting up the configuration for the MySQL container such as the MySQL database configuration, ports configuration, etc.
- Second challenge was to create persistent volumes for the application in order to upload images (profile picture, cover picture, post images) to the server and store the database persistently so that the data is not lost once the container is stopped.
- Third challenge was using CORS. CORS, or Cross-Origin Resource Sharing, is a security feature implemented by web browsers to restrict web pages from making requests to a different domain than the one that served the web page. This security mechanism helps prevent potential security vulnerabilities that could arise from cross-origin requests. There were various errors encountered while using CORS -
    - **Missing CORS Headers:** To enable cross-origin requests, the server must include specific headers in its response, such as Access-Control-Allow-Origin, Access-Control-Allow-Methods, and Access-Control-Allow-Headers. If these headers are missing or incorrect, the browser will block the request.
    - **Wildcard Origin Not Allowed:** If a server is configured to allow requests from specific origins using the Access-Control-Allow-Origin header, using a wildcard (*) may not be allowed. Some servers require the exact origin to be specified.
    - **Credentials and CORS:** When making requests with credentials (such as cookies or HTTP authentication), the server must include the Access-Control-Allow-Credentials header. Additionally, the client must set the withCredentials property to true in the XMLHttpRequest or Fetch API.
    - **Preflight Request Failure:** Certain cross-origin requests trigger a preflight request, which is an HTTP OPTIONS request that checks whether the actual request is safe to send. If the preflight request fails, the actual request is not sent, and an error is reported.
- Fourth challenge was deploying the application to another user on the system using Ansible and SSH. There were various errors faced such as some specific commands

needed to be run using sudo for root permissions and managing the persistent volumes on the deployed host.

Scope of Future Work

The future scope for the project could include the following -

- **Feature Enhancements:**
    - Implementing a robust notification system to keep users informed about relevant events and updates.
    - Integrating real-time chatting functionality to facilitate instant communication among users.
    - Expanding content sharing capabilities to support various multimedia formats, including videos, audio, and other rich media types.
    - Allowing users to make friends and displaying a list of friends and the followers and followings of a user.
- **Monitoring Optimization:**
    - Transition from the current ELK stack log file upload approach to continuous monitoring.
    - Implement real-time monitoring solutions to proactively identify and address performance issues.
- **Deployment Strategy:**
    - Explore and plan for the migration of the application to a cloud service provider. Integrate with cloud-specific services that align with the application's functionality and improve overall performance.
    - Leverage cloud features such as automatic scaling to adapt resources based on demand.
    - Implement load balancing to distribute incoming traffic and ensure optimal performance.
- **User Experience Improvement:**
    - Ensure that the introduction of new features and the transition to a cloud environment contribute positively to the overall user experience.
    - Conduct user testing and gather feedback to refine features and address any usability concerns.

- **Data Handling for Multimedia Content:**
    - Enhance the backend infrastructure to handle the storage and retrieval of multimedia content efficiently.
    - Implement appropriate data compression and optimization techniques to minimize latency in content delivery.
- **Improving Testing**
    - More testing can be done in order to test the functions extensively. Exclusive databases can be created just for the use of testing.
- **Compliance and Security Measures:**
    - Ensure that the application complies with relevant data protection regulations and security standards.
    - Implement robust security measures for data transmission, storage, and access, especially when dealing with sensitive user information.

References

- Class notes and slides
- Calculator Mini Project
- https://carloscuba014.medium.com/building-a-react-app-that-connects-to-mysql-via-nodejs-using-docker-a8acbb0e9788
- https://www.bezkoder.com/react-node-express-mysql/
- https://www.javaguides.net/2023/06/how-to-create-react-app-using-vite.html