

---

CS 816 - Software Production Engineering

# **FINAL PROJECT REPORT**

## Social Media Application

---

Krushikar Reddy - IMT2020043

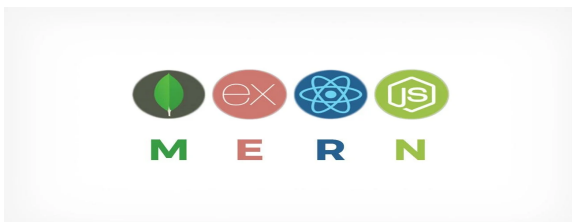
Sougandh Krishna - IMT2020120

---

## **Project Description**

The social media application - ExpressHub is a platform where people can connect online. It's a digital space where users can share posts, and pictures and interact with their friends. The main goal is to create a user-friendly platform for meaningful connections and enjoyable social experiences.

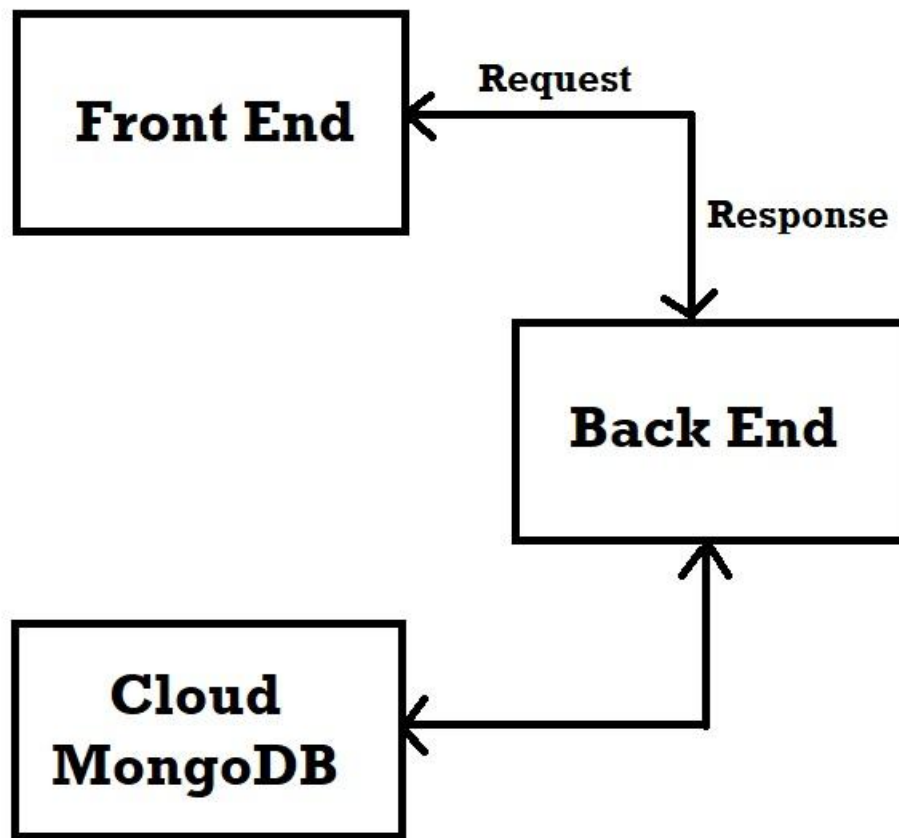
## **Tech Stack Used (MERN)**



- MongoDB: A flexible and scalable NoSQL database. Ideal for handling diverse and evolving data structures.
- Express: A minimalist and flexible framework that simplifies building robust and scalable backend (server-side) applications.
- React: A powerful and declarative JavaScript library for building user interfaces, enabling the creation of dynamic and responsive front-end applications.
- Node: A JavaScript runtime that executes server-side script effectively, making it a key component in building scalable and high-performance web applications.

---

## Architecture

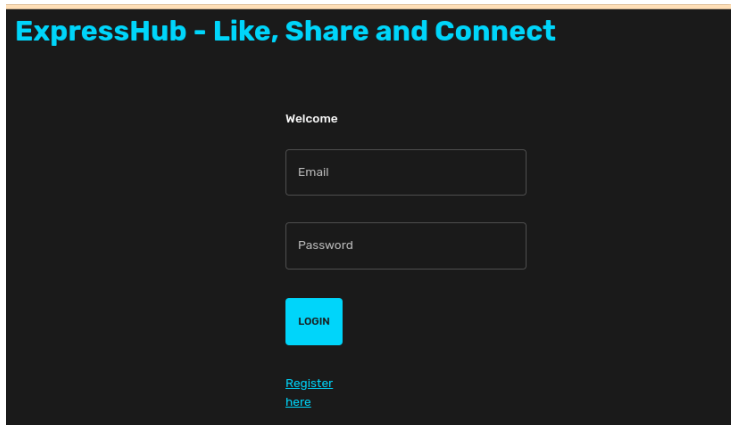


---

## Features

- 1) **Register and Login Page:** The register page allows users to create an account in the application, with appropriate details. The users can enter the application through the login page.

### Login Page



ExpressHub - Like, Share and Connect

Welcome

Email

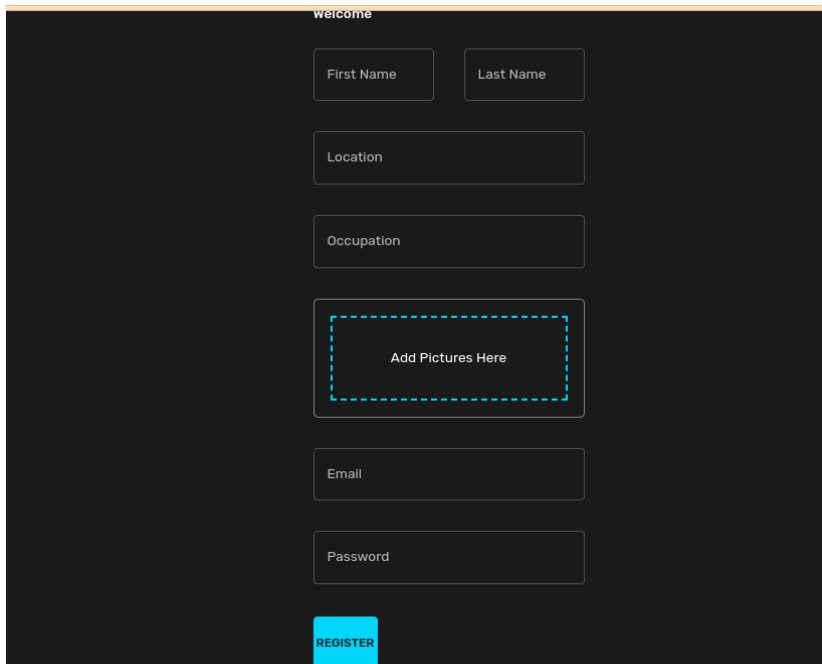
Password

LOGIN

[Register here](#)

The login page features a dark background with a light blue header. It includes a 'Welcome' message, two input fields for 'Email' and 'Password', a blue 'LOGIN' button, and a link to 'Register here'.

### Register Page



welcome

First Name

Last Name

Location

Occupation

Add Pictures Here

Email

Password

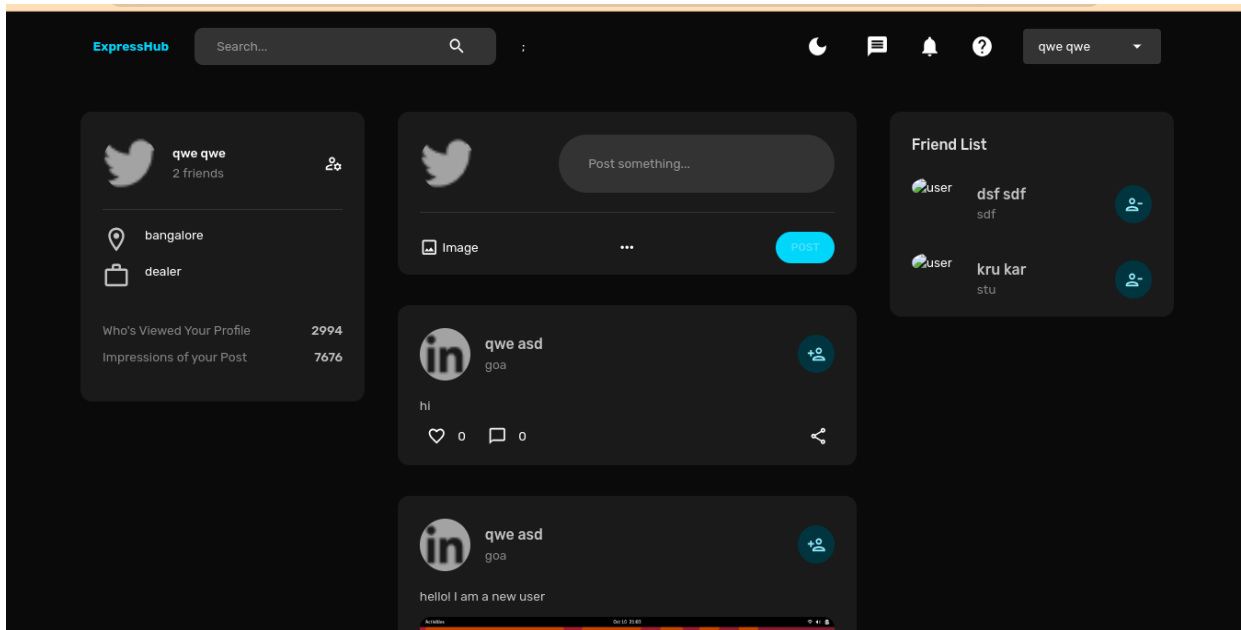
REGISTER

The register page has a dark background with a light blue header. It includes a 'welcome' message, two input fields for 'First Name' and 'Last Name', a 'Location' input field, an 'Occupation' input field, a dashed box for 'Add Pictures Here', an 'Email' input field, a 'Password' input field, and a blue 'REGISTER' button.

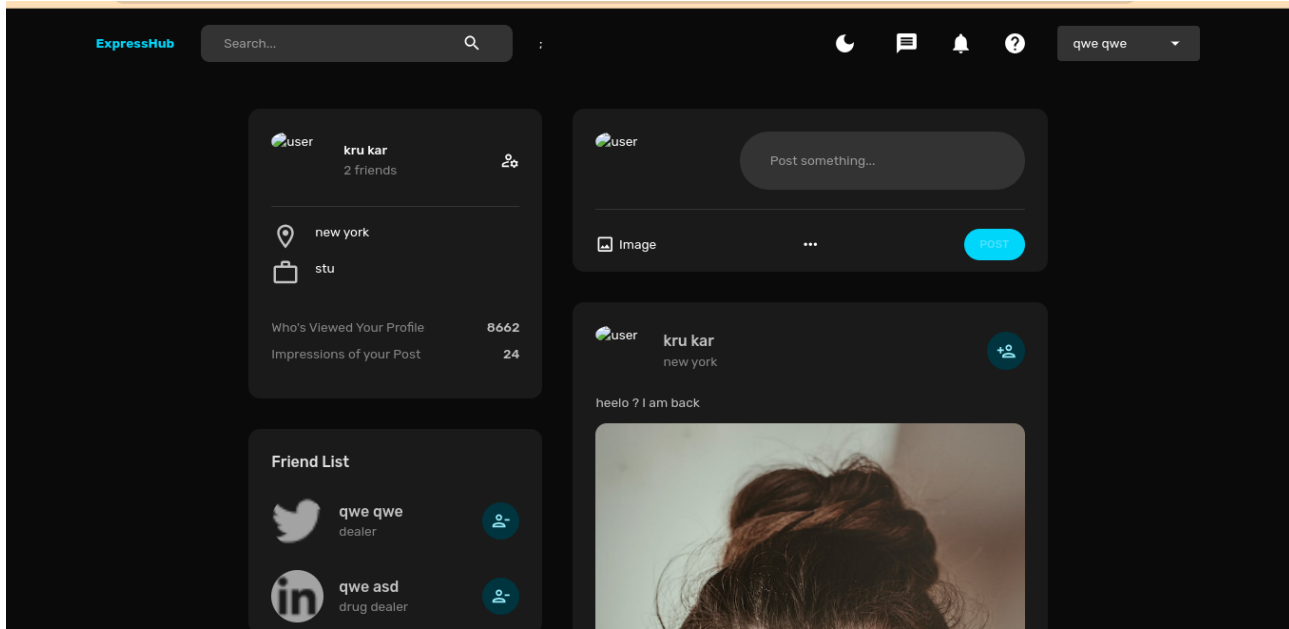
---

## 2) **Home Page:** The Home Page contains the following :

- a. **Navbar:** The Navigation bar essentially allows the user to return to the home page, Toggle between white and dark background, and log out.
- b. **User Info:** Information of the user as provided during registration.
- c. **Posts section:** This section allows the user to share their thoughts and images. It also contains a list of all the Posts from all users. Users can list, comment on posts, and add friends.
- d. **Friends List:** This section contains the list of all the friends of the user.



**3) User Page:** This page is specific to a user. It contains all the interactions of the user in this application. It contains the basic information about the user, the user's posts, and the user's friends list.



## API Documentation

- User Routes:

```
const router = express.Router();

// READ
router.get("/:id", verifytoken, getUser);
router.get("/:id/friends", verifytoken, getUserFriends);

// UPDATE
router.patch("/:id/:friendId", verifytoken, addRemoveFriend);
```

---

The first two routes involve the GET request. The first route fetches the particular user from the database based on the id. The second route fetches the list of friends of a particular user based on user id.

The third route involves the PATCH request. It involves the addition/removal of friend users from the friends list of a particular user.

- Post Routes :

```
const router = express.Router();

router.get("/", verifytoken, getFeedPosts);
router.get("/:userId/posts", verifytoken, getUserPosts);

router.patch("/:id/like", verifytoken, likePost);
export default router;
```

The first two routes involve GET requests which fetch all the posts and user-specific posts from the MongoDB, respectively. The third route involves a PATCH request, which adds/removes a user from the list of liked users for a post.

- Login Route:

```
5
6   router.post("/login", login);
7
```

This is the login route. At this endpoint, the Authentication of a user happens. The login function checks if the email is in the database, verifies the password, and reacts accordingly. It will either

---

successfully allow the user to log in, indicate if the credentials are invalid (password is wrong), or notify if the user doesn't exist.

- Routes Involving Files :

```
// ROUTES WITH FILES

app.post("/auth/register", upload.single("picture"), register);
app.post("/posts", verifytoken, upload.single("picture"), createPost);
```

There are two places where the user would probably upload files, once while registering (for profile icon) and other times while uploading an image in the post. Therefore, these routes are defined separately. Both of these involve POST requests, where the data from the request body is posted on the MongoDB. The picture path is stored in MongoDB, while the actual picture is stored locally in the public/assets folder.

In all the above routes (excluding the login and register), there is a verify token function which does the authorization of users, based on a token. Only the Authorized Users will be able to perform operations from these routes.

## **Repositories**

Github: [https://github.com/krushikar543/SPE\\_finalProj-Social-Media-App](https://github.com/krushikar543/SPE_finalProj-Social-Media-App)



---

## **DevOps Tools used**

- **Version Control(Git, GitHub)**

Implementing version control ensures that you can track changes to your code over time. It helps us maintain a history of our code and collaborate with others effectively.

Git allows us to track changes in the codebase, collaborate seamlessly, and easily roll back changes if needed. It ensures a smooth workflow for multiple developers working on different features simultaneously.

GitHub, a platform that hosts Git repositories, complements this by providing a collaborative space for sharing code, managing projects, and facilitating features like pull requests and issue tracking.

- **Continuous Integration / Continuous Deployment (CI/CD) (Jenkins)**

A CI/CD pipeline helps you automatically build, test, and deploy your code. This can be useful for any project, as it ensures that changes are continuously integrated and tested, reducing the risk of introducing bugs.

We used Jenkins for the establishment of the CI/CD pipeline. It is an open-source automation server that facilitates continuous integration and continuous delivery (CI/CD) in software development. This allowed automated building, testing, and deployment of our web

---

application, ensuring that new features and updates could be delivered to users rapidly.

- Containerization (Docker)

Containerization simplifies deployment and ensures that the application runs consistently across different environments. We used Docker to containerize the application.

Docker is a platform that empowers developers to build, deploy, and execute applications within lightweight, portable, and self-contained containers. These containers encapsulate applications and dependencies, ensuring consistent functionality across various computing environments. Docker simplifies the complexities of software deployment and administration, streamlining processes for constructing and sharing applications.

In the context of our application, Docker containers encapsulate both frontend and backend components, ensuring consistency across different development environments and simplifying the deployment process. This portability enables the application to run reliably on various systems, from developers' local machines to production servers. Docker's efficiency in resource utilization facilitates easy scaling for handling increased loads as the platform grows. The isolation of services within containers promotes modularity, allowing independent development and scaling of the front end and back end. Furthermore, Docker adoption streamlines the continuous integration

---

and continuous deployment (CI/CD) pipeline, fostering automation, reliability, and efficient dependency management.

- Infrastructure as Code (Ansible)

Ansible is an open-source automation tool that allows you to deploy, configure, and manage systems and applications across multiple servers. This ensures that you have a reproducible and consistent setup, which is valuable when collaborating with others or moving the project to different machines. It uses a declarative language called YAML to define and automate tasks, making it easy to set up and maintain systems, manage software installations, and orchestrate complex workflows across servers and infrastructure components.

- Multi-Container Orchestration (Docker compose)

Docker Compose is a DevOps tool that simplifies the management of multi-container Docker applications. It allows you to define and manage multiple containers, their configurations, and the services they depend on, all in a single file called `docker-compose.yml`. With Docker Compose, you can define an entire application stack, including networks, volumes, and services, and then use a single command to spin up the entire environment.

In the context of our project, Docker Compose allowed us to define, configure, and manage both frontend and backend containers in a single `docker-compose.yml` file. This simplifies the orchestration of

---

the entire application stack, making it easy to start, stop, and manage multiple containers with a single command.

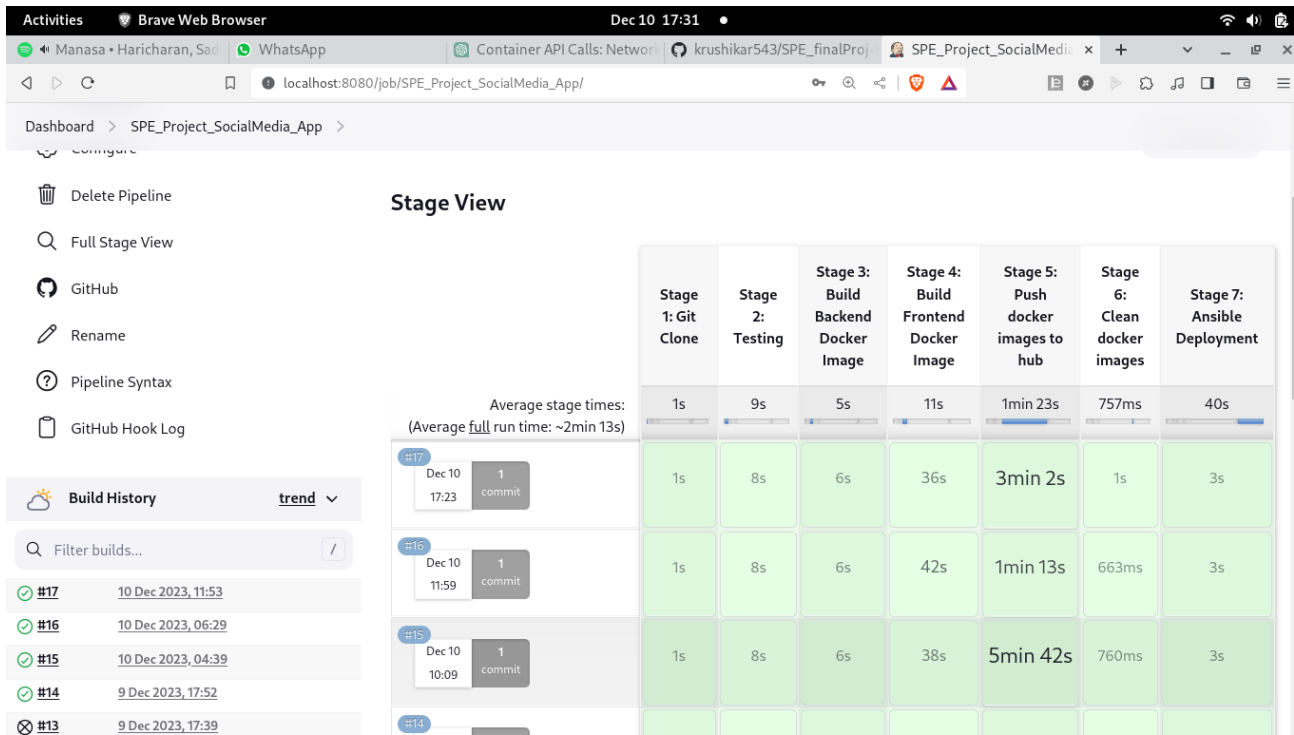
- Logging and Monitoring (ELK stack)

The ELK stack (Elastic Search, Logstash, and Kibana) is widely used for centralized logging and real-time monitoring, offering a comprehensive solution for analyzing and visualizing logs using the log file that is created.

In the context of our project, Elasticsearch receives index logs from both the frontend and backend containers. It allows us to perform complex queries on the logs for analysis. Logstash collects logs from the containers, parses them, and sends them to Elasticsearch. It ensures that logs are structured and standardized for better analysis. And then Kibana is the visualization tool that allows us to explore, visualize, and interact with the logs stored in Elasticsearch.

So ELK stack provides us with a powerful solution for centralized log management, real-time monitoring, and insightful analytics. It also enhances our ability to proactively address issues, optimize performance, and ensure a seamless and reliable user experience on the social media platform.

# Jenkins Pipeline Stages

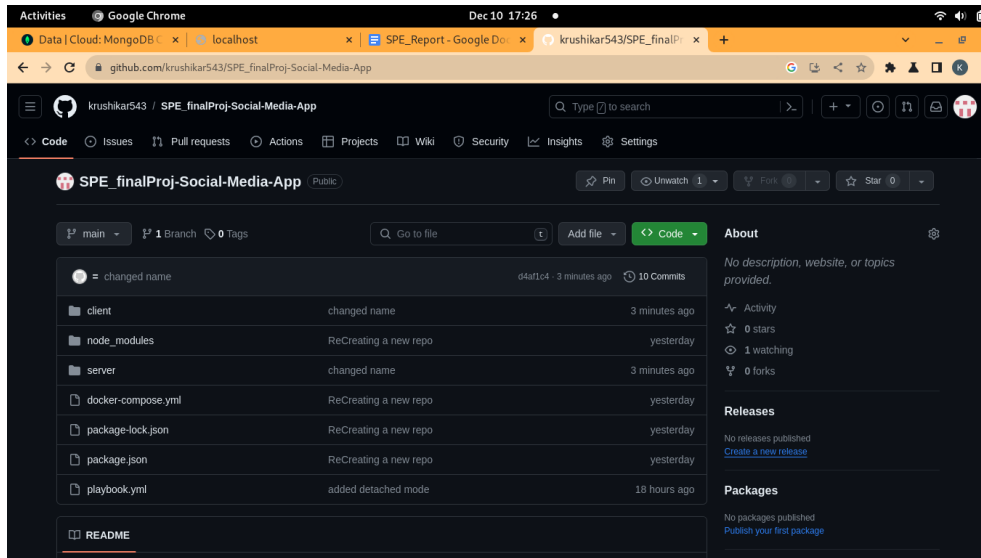


There are a total of 7 stages in My Jenkins Pipeline :

1. Git Clone Stage: At this stage, Jenkins clones the GitHub repository locally.

```
10 stage('Stage 1: Git Clone'){
11     steps{
12         git branch: 'main',
13             url: 'https://github.com/krushikar543/SPE_finalProj-Social-Media-App.git'
14     }
15 }
```

## GitHub Repository:



2. Testing Stage: In this stage, We have done unit testing using Mocha and Chai for the login authentication.

```
16 stage('Stage 2: Testing'){
17     steps{
18         dir('server'){
19             sh "npm test"
20         }
21     }
22 }
```

Mocha and Chai are popular JavaScript testing frameworks used together to create and run test suites, including for authentication in login processes. It provides us with a robust and organized approach to login authentication.

---

Mocha is a versatile testing framework that provides a structure for organizing and running test cases. It supports various testing styles (such as BDD and TDD) and allows for asynchronous testing.

Chai is an assertion library that works seamlessly with Mocha. It provides a set of assertion styles to make your test cases more readable and expressive.

We used Mocha to structure our authentication tests and combined Chai with Mocha to write clear and concise assertions in our test cases (Verify that valid login credentials etc.)

3. Build Backend Docker Image: In this stage, Jenkins builds the backend (server-side) image.

```
23 ▾ stage('Stage 3: Build Backend Docker Image'){
24 ▾   steps {
25 ▾     script {
26 ▾       dir('server'){
27         backend_image = docker.build "krushikar/backend_image:latest"
28       }
29     }
30   }
31 }
```

The Dockerfile is as follows:

```
server > dockerfile > ...
1 FROM node:18.13.0
2 WORKDIR /
3 COPY package*.json ./
4 COPY . .
5 EXPOSE 3001
6 CMD ["npm", "start"]
```

- FROM node:18.13.0: Specifies the base image for the container, and uses Node.js version 18.13.0 as the runtime environment.
- WORKDIR /: Sets the working directory within the container to the root directory.
- COPY package\*.json ./: Copies package.json and package-lock.json to the container's current working directory.
- COPY . .: Copies all the files to the container's current working directory.
- EXPOSE 3001: Inform Docker that the application will use port 3001, allowing external access to that port.
- CMD ["npm", "start"]: Defines the command to run when the container starts, initiating the application.

4. Build FrontEnd Docker Image: In this stage, Jenkins builds the frontend (client-side) docker image.

```
32 stage('Stage 4: Build Frontend Docker Image'){
33     steps {
34         script {
35             dir('client'){
36                 frontend_image = docker.build "krushikar/frontend_image:latest"
37             }
38         }
39     }
40 }
```



---

The Dockerfile is as follows:

```
client > 🐳 dockerfile > ...  
1 FROM node:18.13.0  
2 WORKDIR /  
3 COPY package*.json ./  
4 COPY . .  
5 RUN npm install  
6 EXPOSE 3000  
7 CMD ["npm", "start"]
```

- FROM node:18.13.0: Specifies the base image for the container, and uses Node.js version 18.13.0 as the runtime environment.
- WORKDIR /: Sets the working directory within the container to the root directory.
- COPY package\*.json ./: Copies package.json and package-lock.json to the container's current working directory.
- COPY . . : Copies all the files to the container's current working directory.
- RUN npm install: This command installs all the dependencies required to run the application, based on the package.json file.
- EXPOSE 3000: Informs Docker that the application will use port 3000, allowing external access to that port.
- CMD ["npm", "start"]: Defines the command to run when the container starts, initiating the application.

5. Pushing Docker Images to Docker Hub: After building the docker images, Jenkins pushes the images into the docker hub.

```

41 ▾ stage('Stage 5: Push docker images to hub') {
42 ▾     steps {
43 ▾         script {
44 ▾             docker.withRegistry('', 'DockerHubCred') {
45 ▾                 backend_image.push()
46 ▾                 frontend_image.push()
47 ▾             }
48 ▾         }
49 ▾     }
50 ▾ }

```

## 6. Clean docker Images:

This stage is intended to clean up any unused Docker containers and Images, forcefully without confirmation prompts.

```

51 ▾ stage('Stage 6: Clean docker images'){
52 ▾     steps{
53 ▾         script{
54 ▾             sh 'docker container prune -f'
55 ▾             sh 'docker image prune -f'
56 ▾         }
57 ▾     }
58 ▾ }

```

## 7. Ansible Deployment:

In this stage, Jenkins runs the ansible-playbook, which deploys the application on the specified hosts.

```

59 ▾ stage('Stage 7: Ansible Deployment'){
60 ▾     steps{
61 ▾         script{
62 ▾             sh 'ansible-playbook -i inventory playbook.yml'
63 ▾         }
64 ▾     }
65 ▾ }

```

## Ansible Playbook (yaml) file :

```
! playbook.yml X
! playbook.yml > {} 0 > [ ]tasks > {} 0 > command
Ansible Playbook - Ansible playbook files (ansible.json)
1 ---
2 - name : Deploy Social-Media-App
3   hosts: localhost
4
5   tasks :
6     # - name: Copy Docker Compose file
7     #   copy:
8     #     src: docker-compose.yml
9     #     dest: "{{ playbook_dir }}/docker-compose.yml"
10
11   - name : Run Docker Compose
12     command: docker-compose up -d
```

- The commented part of the code is used to specify to copy the docker-compose file into other hosts for deployment.
- Then we run the docker compose file.

## Docker Compose (yaml) file :

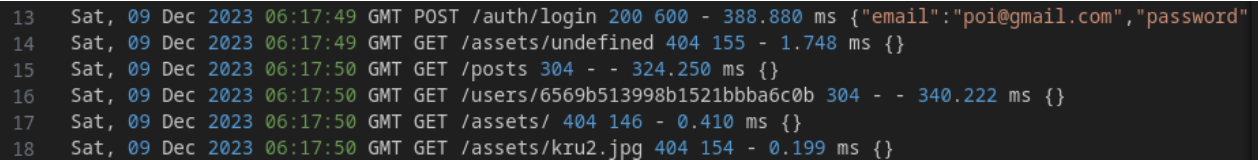
```
🔥 docker-compose.yml X
🔥 docker-compose.yml > {} services > {} backend > [ ] environment > 1
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)
1 version : '3'
2 services :
3   frontend :
4     image : krushikar/frontend_image:latest
5     ports :
6       - '3000:3000'
7     depends_on :
8       - backend
9   backend :
10    image : krushikar/backend_image:latest
11    ports :
12      - '3001:3001'
13    environment :
14      - MONGO=mongodb+srv://krushikar:LSncGHKQ1WHhjvNc@cluster0.3xj17bx.mongodb.net/?retryWrites=true&w
15      - JWT=somesecretkeyveryhardtocrack
```

- 
- “Services” defines the services that compose your application.  
In this file, there are two services: frontend and backend
  - frontend:
    - Specifies the Docker image to use for the frontend service.
    - Maps port 3000 on the host to port 3000 on the container for the frontend service. This allows you to access the frontend application externally on port 3000.
    - Also indicates that the frontend service depends on the backend service. This ensures that Docker Compose starts the backend service before the frontend service.
  - backend:
    - Specifies the Docker image to use for the backend service.
    - Maps port 3001 on the host to port 3001 on the container for the backend service. This allows you to access the backend application externally on port 3001.
    - Sets environment variables for the backend service. It provides connection details for MongoDB using the MONGO variable and a secret key for JWT (JSON Web

---

Token) using the JWT variable. The actual values for these variables are provided as placeholders in this example.

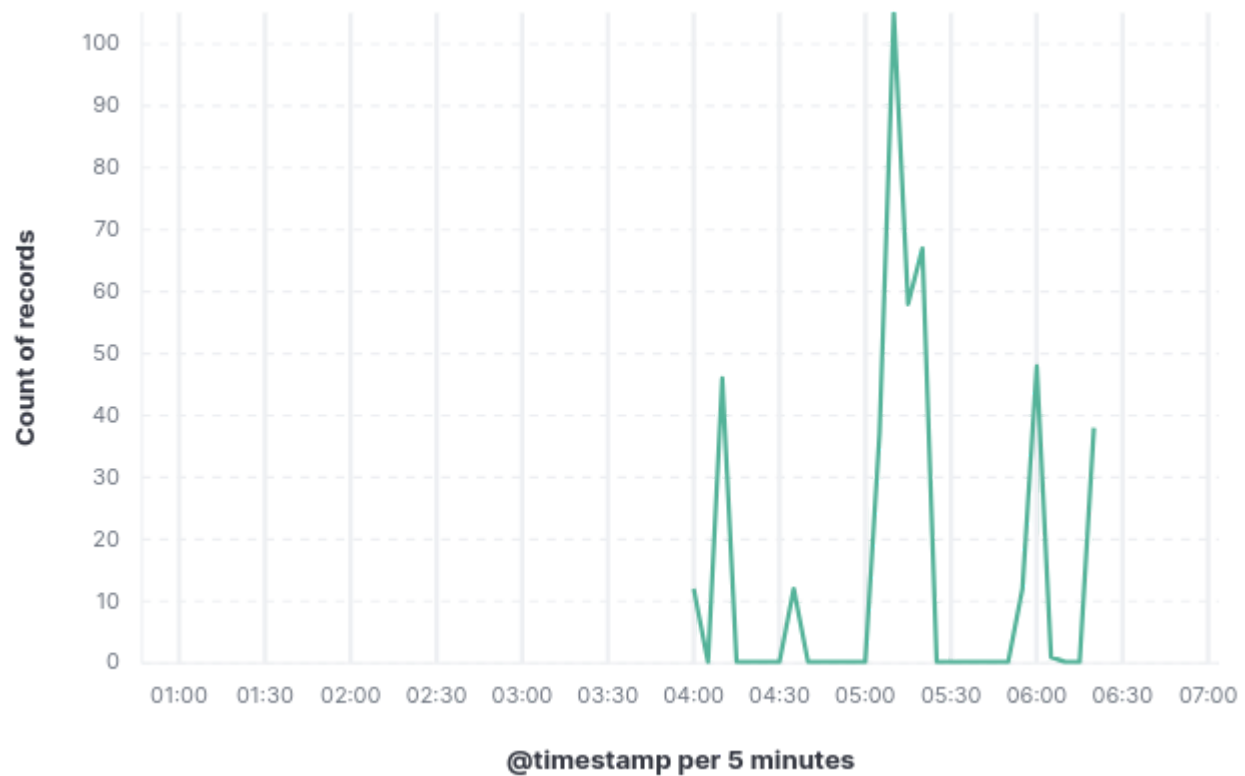
## **Logs and Kibana Visualisations**



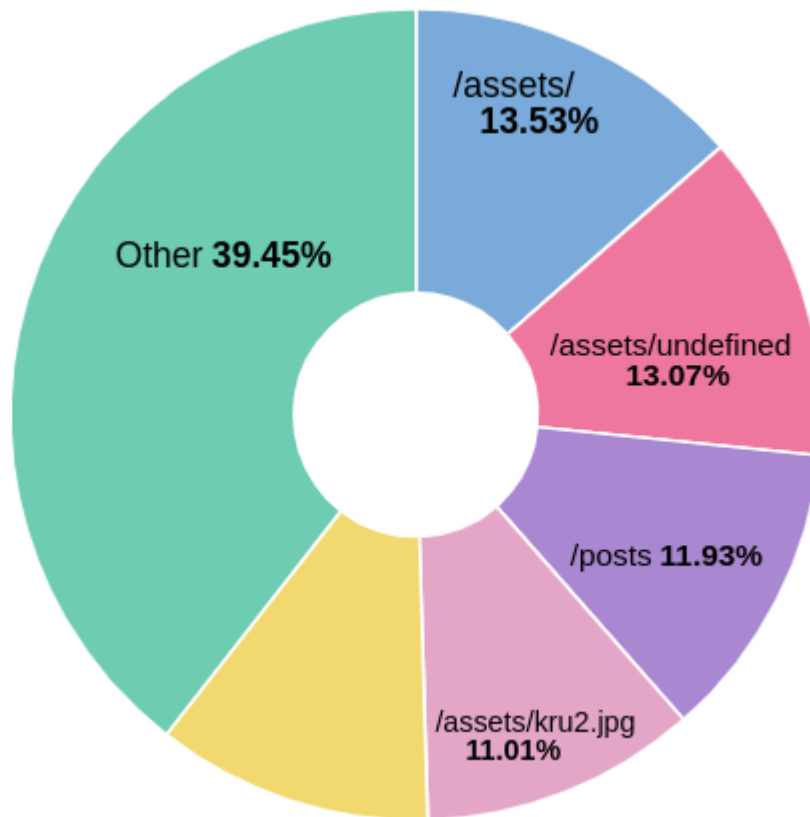
```
13 Sat, 09 Dec 2023 06:17:49 GMT POST /auth/login 200 600 - 388.880 ms {"email":"poi@gmail.com","password"
14 Sat, 09 Dec 2023 06:17:49 GMT GET /assets/undefined 404 155 - 1.748 ms {}
15 Sat, 09 Dec 2023 06:17:50 GMT GET /posts 304 - - 324.250 ms {}
16 Sat, 09 Dec 2023 06:17:50 GMT GET /users/6569b513998b1521bbba6c0b 304 - - 340.222 ms {}
17 Sat, 09 Dec 2023 06:17:50 GMT GET /assets/ 404 146 - 0.410 ms {}
18 Sat, 09 Dec 2023 06:17:50 GMT GET /assets/kru2.jpg 404 154 - 0.199 ms {}
```

This is a sample screenshot from the log file generated. Here we have 6 fields, representing Time Stamp [Day, Date, Time(GMT)], Request Type, End Point accessed, Backend response, Time taken, Authentication details.

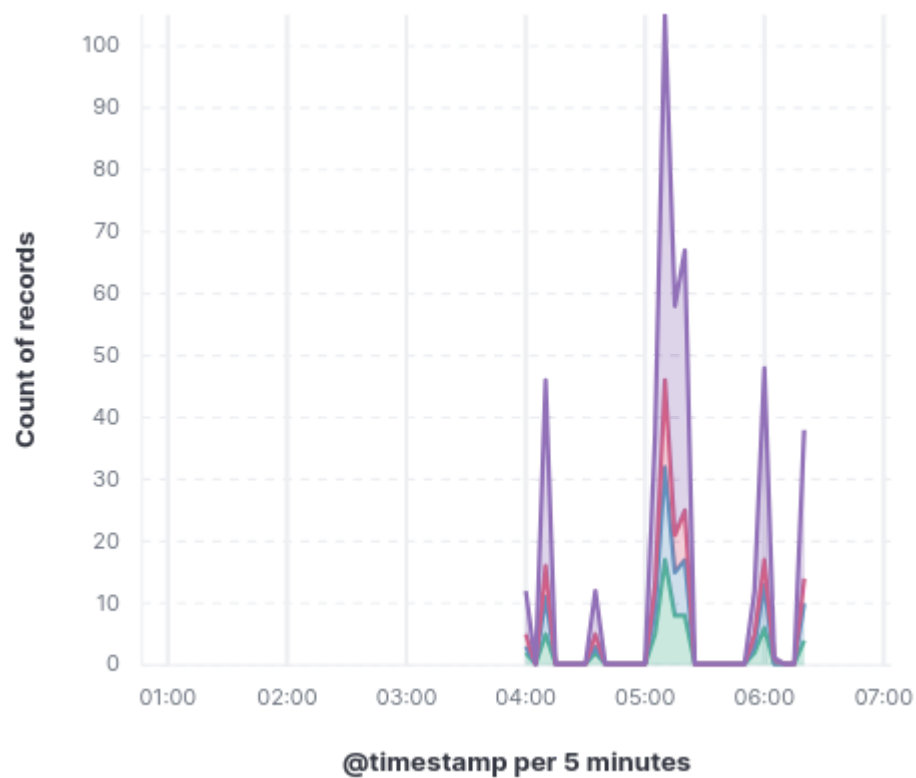
Following visualizations were created using Kibana by uploading the log files:



The above line plot illustrates the number of interactions recorded within the application during the specified time frame on December 10th, with timestamps presented in GMT.



This donut chart shows the end points accessed during the same timeframe on 10th December.



This area chart shows the number of interactions and end points accessed during the same time frame on December 10th, in a single plot.