SPE Major Project Report

IMT2021509 N.V.S.Asrith
IMT2021081 K.Gowtham
22BTRCL171 Y.PavanReddy

Project Overview:

For our major project, we developed a **Real-Time Chat Application** using the **MERN Stack** (MongoDB, Express, React, and Node.js). The application is fully deployed with **Docker** and **Kubernetes**, ensuring seamless scalability and deployment.

Frontend

The frontend is built using **React.js**, a powerful JavaScript library for building dynamic user interfaces. We employed **Redux Toolkit** to manage the state effectively, ensuring smooth user interaction, especially in real-time communication. The user interface is styled using **Tailwind CSS**, which allows us to create a responsive and modern design with minimal effort. Key frontend features include:

- **Login and Registration:** Users can sign up using their email address and log in to the platform to access their chats.
- **Real-Time Chat:** The application allows users to send and receive messages in real-time, creating an engaging and interactive experience.
- **Profile Management:** Users can update their profile picture and personal details.
- **Image Sharing:** Users are able to send images during conversations, enhancing the chat experience.

The user experience is optimized to be intuitive and user-friendly, with interactive elements and a clean design.

Backend:

The backend is powered by **Node.js** and **Express.js**, enabling efficient handling of real-time interactions. The backend handles:

- **User Authentication:** The system uses email-based registration and login, ensuring secure access to the application.
- **Real-Time Communication:** We leveraged **Socket.io** to handle real-time messaging between users, providing instant updates without needing to refresh the page.
- **Data Storage: MongoDB** is used to store user data and messages in a NoSQL format, allowing flexible and scalable storage solutions.
- **Profile Management and Image Storage:** Profile images and other user data are stored in the backend, with support for image uploads.

Key Technologies Used:

- **Frontend:**
    - React.js
    - Redux Toolkit
    - Tailwind CSS
    - Socket.io (for real-time communication)
- **Backend:**
    - Node.js
    - Express.js
    - MongoDB
    - Socket.io (for real-time communication)
- **Deployment:**
    - Docker
    - Kubernetes

How to Run the Application Locally:

To run the application locally, follow these steps:

1. **Start the Backend (Server):**
    a. Open a terminal and navigate to the **server** directory of the project.
    b. Run the following command to start the server:

```
npm run dev
```

or alternatively:

```
npm run start
```

This will start the backend server, which handles all API requests and real-time messaging.

2. **Start the Frontend (Client):**
    a. Open another terminal and navigate to the **client** directory.
    b. Run the following command to start the React application:

```
npm start
```

This will start the React development server and open the application in the browser (usually at http://localhost:3000).

With both the frontend and backend running, the chat application will be fully operational, with real-time communication between users.

Deployment with Docker and Kubernetes (Minikube):

For the deployment of the chat application, we utilized **Docker** and **Kubernetes** to ensure a scalable and efficient deployment process. Specifically, we used **Minikube** to run Kubernetes locally for development and testing.

Docker:

- We containerized both the frontend and backend of the application using Docker. Docker allows us to package the entire application, including all dependencies, into containers. These containers ensure consistency across different environments (development, testing, production). Both the frontend (React.js) and backend (Node.js/Express.js) were placed into separate Docker containers for easy management and isolation.
    - Docker images were created for both parts of the application, making them portable and allowing for isolated execution across different systems.

Kubernetes with Minikube:

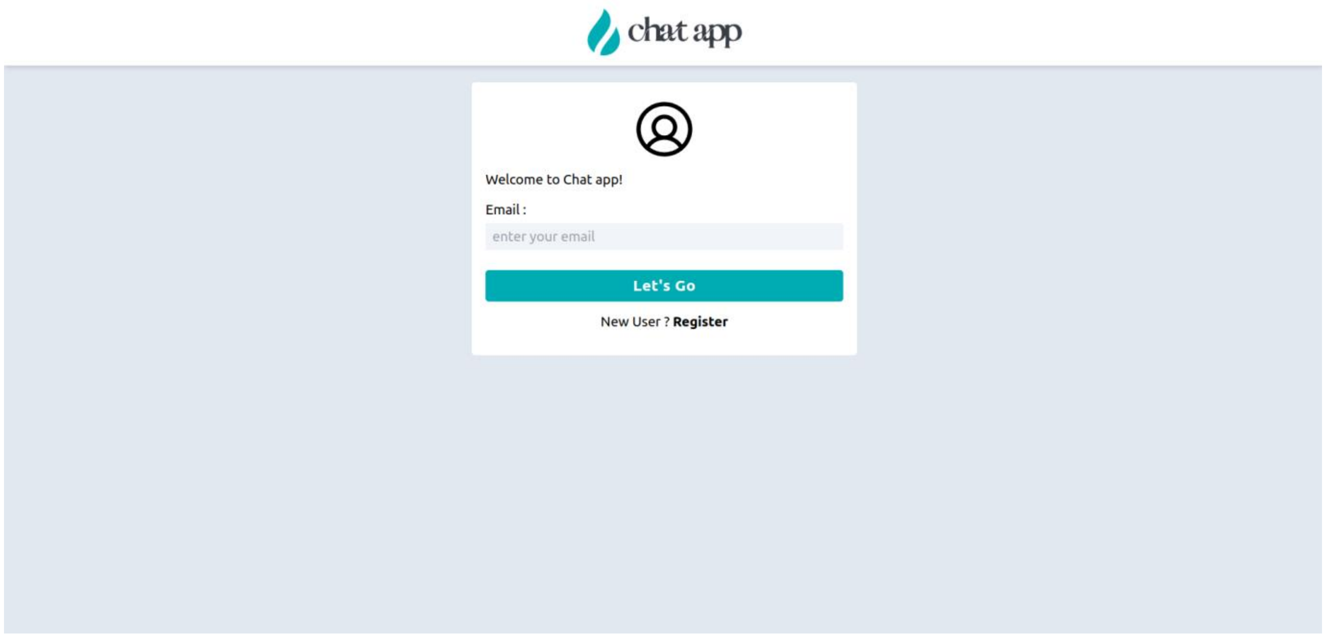
To manage and orchestrate the containers locally, we used **Minikube**, a tool that creates a local Kubernetes cluster. Minikube is perfect for development environments, providing a single-node Kubernetes cluster for testing and running the application.

- **Auto-Scaling**: Minikube helps in setting up auto-scaling for the application, adjusting the number of running containers based on demand.
- **Load Balancing**: Minikube allows us to simulate how Kubernetes will distribute traffic across multiple pods (containers) for load balancing.
- **High Availability**: Minikube also supports the creation of multiple container replicas to ensure high availability, simulating production environments where redundancy is critical.
- **Seamless Deployment**: Minikube makes it easy to deploy new versions of the application, with simple Kubernetes commands to manage updates and rollbacks with minimal downtime.

By using **Docker** and **Minikube**, we ensured a reliable local development environment for testing the scalability and deployment capabilities of our application before moving it to a production environment.

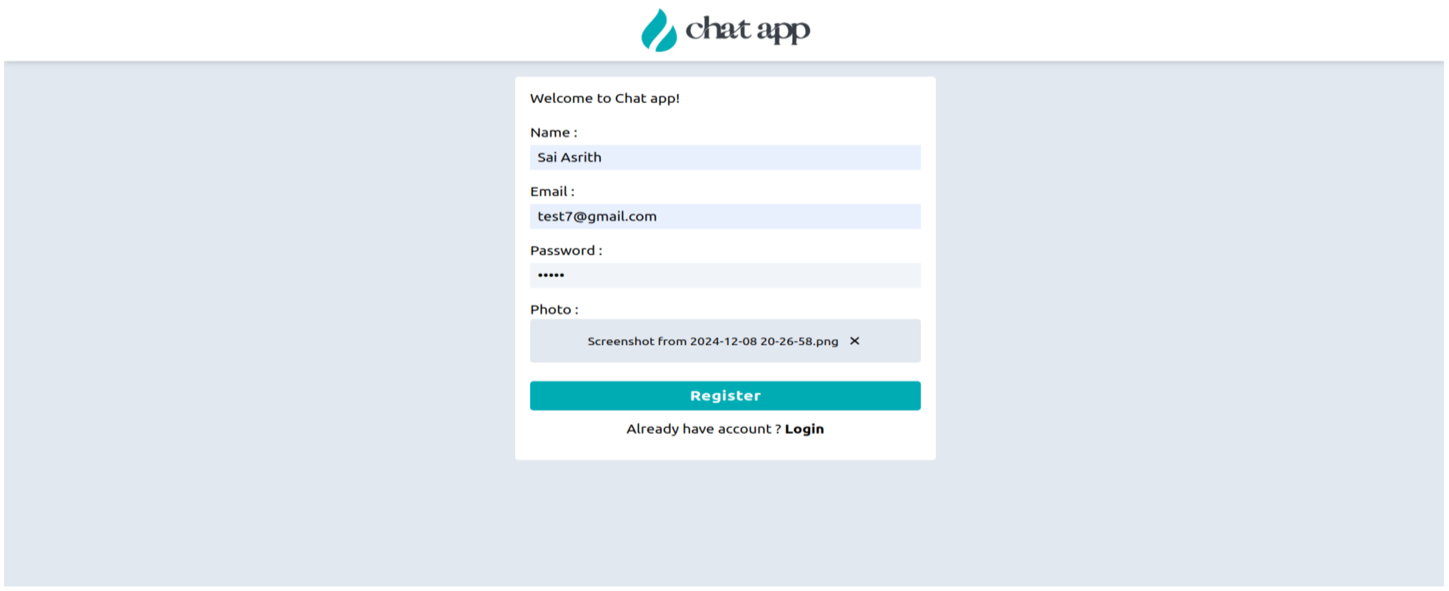
View of the Website

The default page is the **Email Page**. If you want to create a new user, click on **Register** to create an account, or if you already have an account, you can log in using your email.



Register Page:

On the **Register Page**, users can create an account by entering their **email address** and **password**. After submitting the details, the user is redirected to the **Email Page** to log in with the newly created credentials.



chat app

Welcome to Chat app!

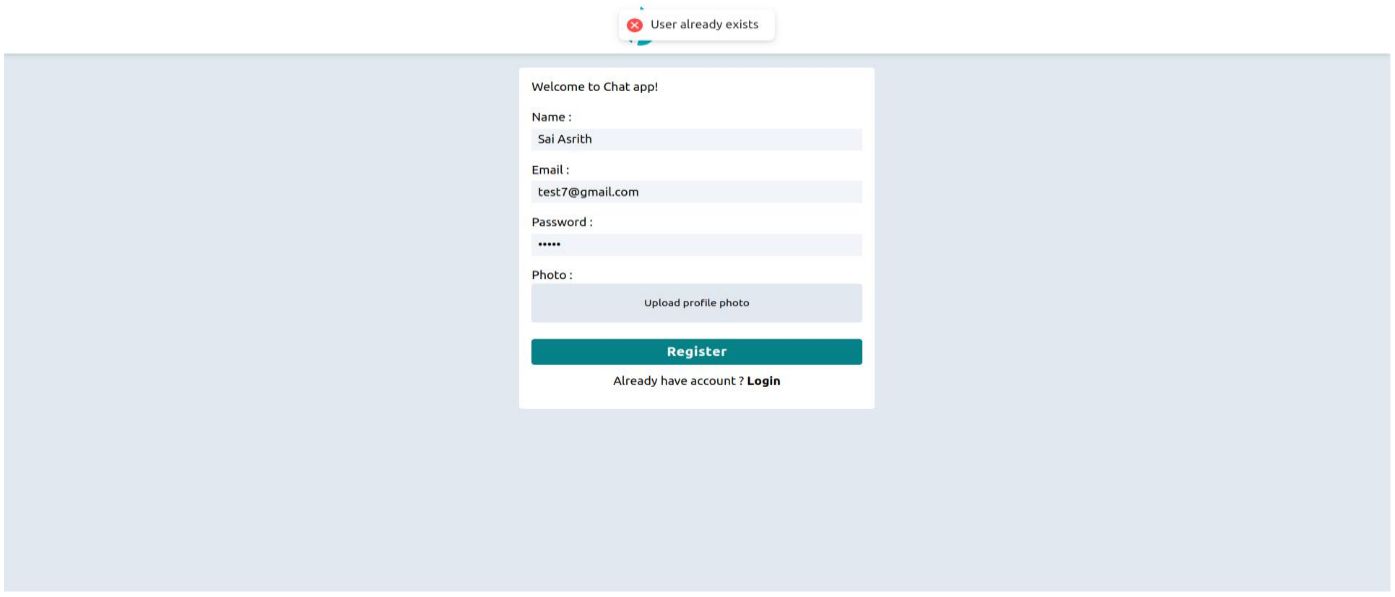
Name :
Sai Asrith

Email :
test7@gmail.com

Password :
•••••

Photo :
Screenshot from 2024-12-08 20-26-58.png ✕

If the user is already registered, you will see output like this:



User already exists

Welcome to Chat app!

Name :
Sai Asrith

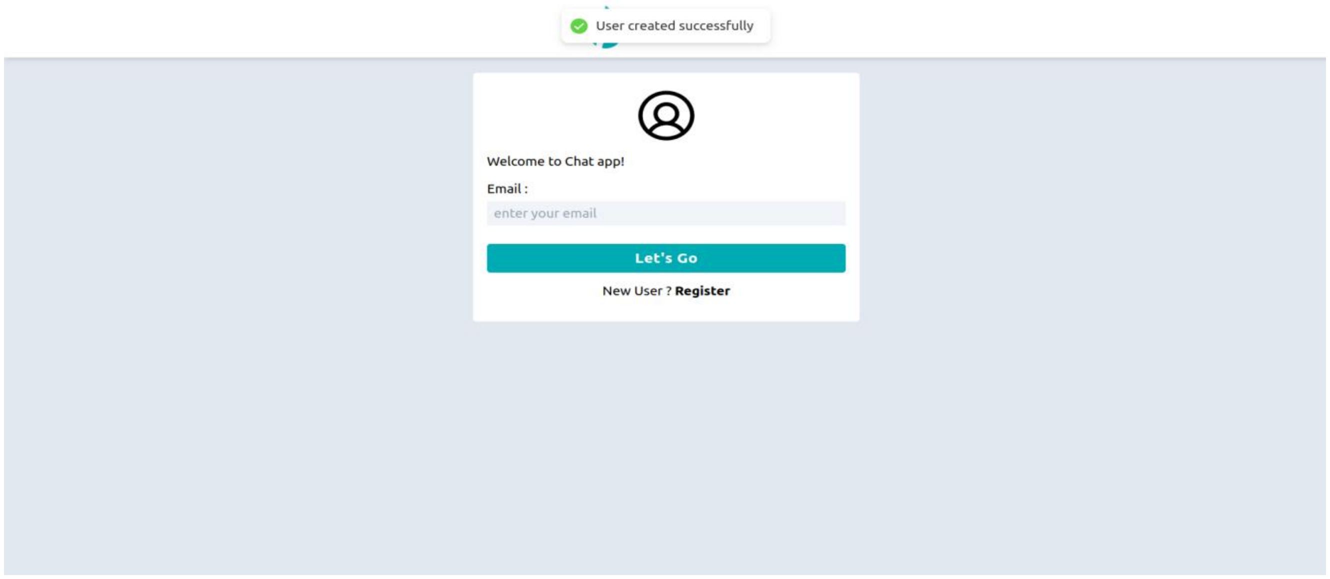
Email :
test7@gmail.com

Password :
•••••

Photo :

<u>Email Verification Page</u>

The **Email Verification Page** takes the user's **email address** as input. If the entered email is correct, the user is redirected to the **Check Password Page**. If the email is incorrect, the page prompts the user to check and enter the correct email.

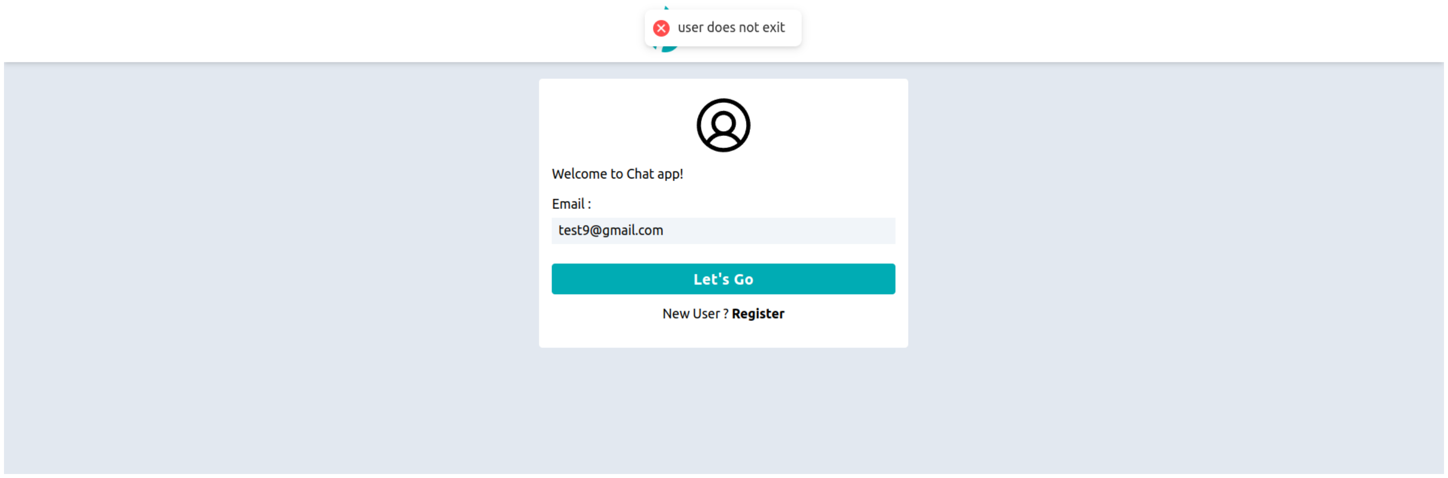


User created successfully

Welcome to Chat app!

Email :

enter your email



user does not exit

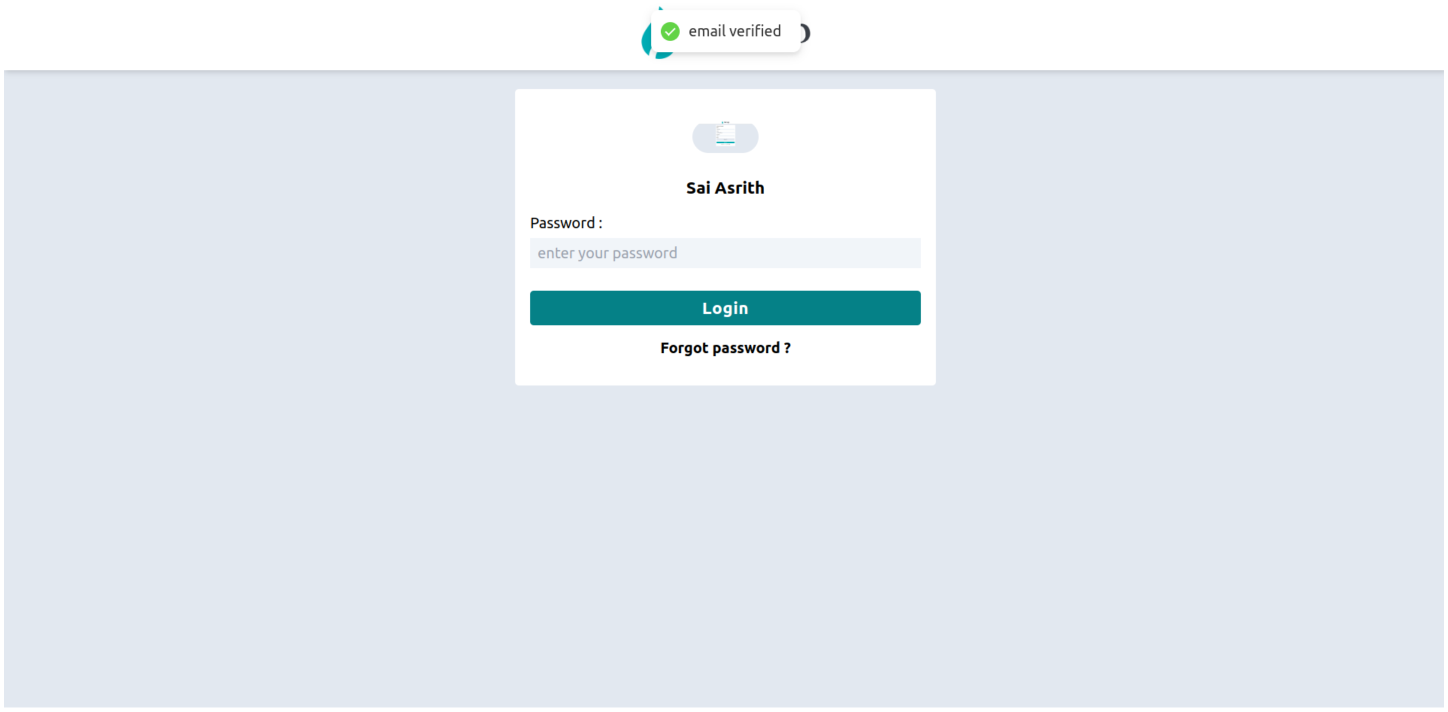
Welcome to Chat app!

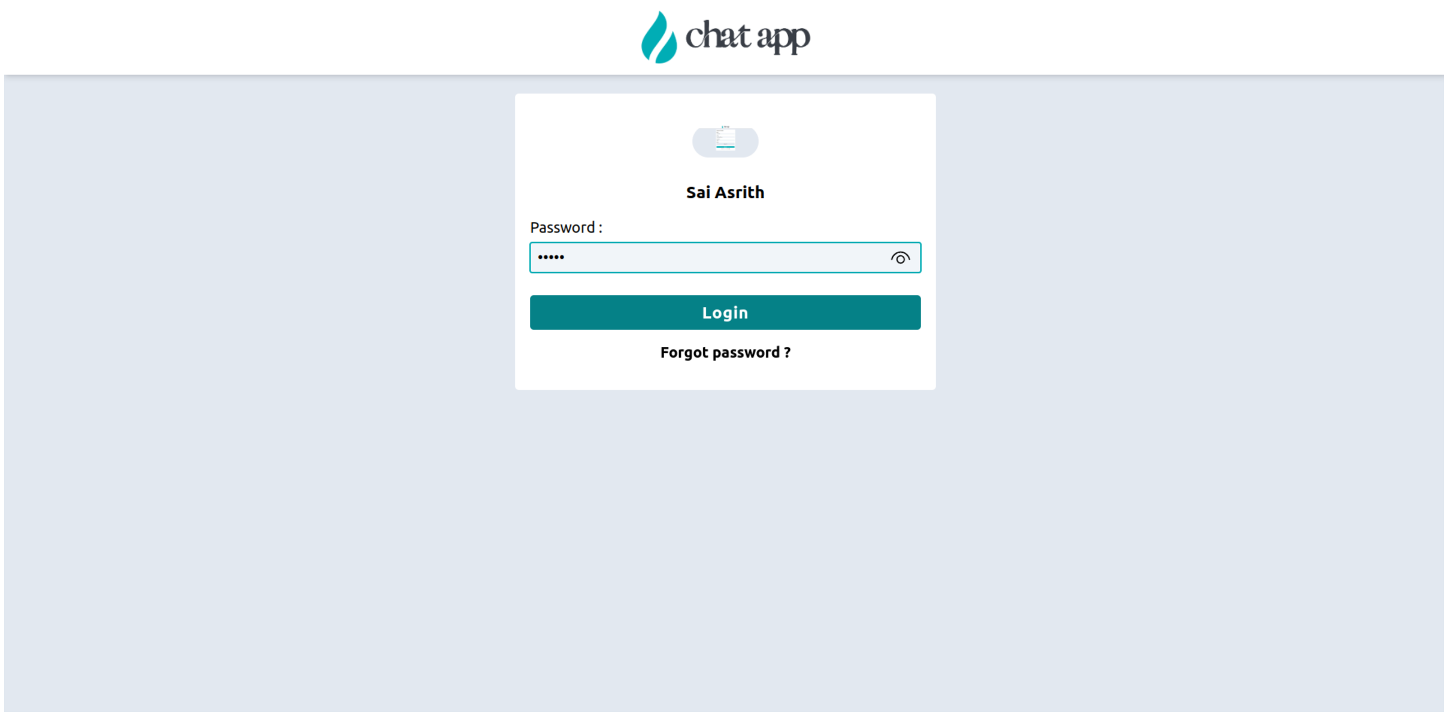
Email :

test9@gmail.com

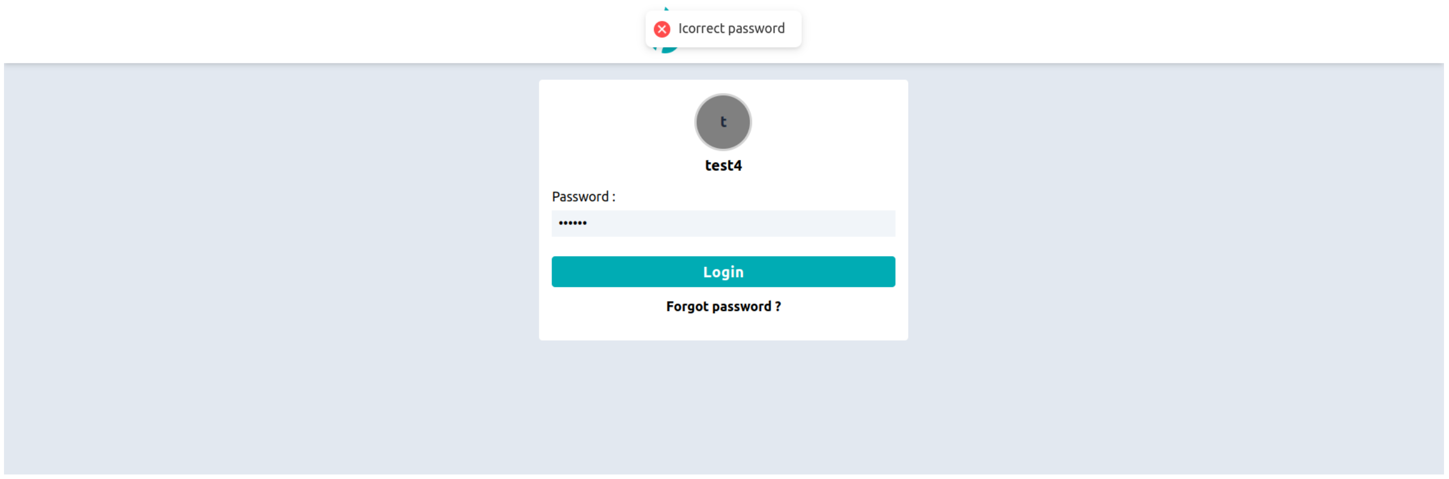
Check Password Page

The **Check Password Page** asks the user to enter their **password**. If the entered password is correct, the user is redirected to the home page. If the password is incorrect, an error message is displayed, asking the user to check the password and try again.





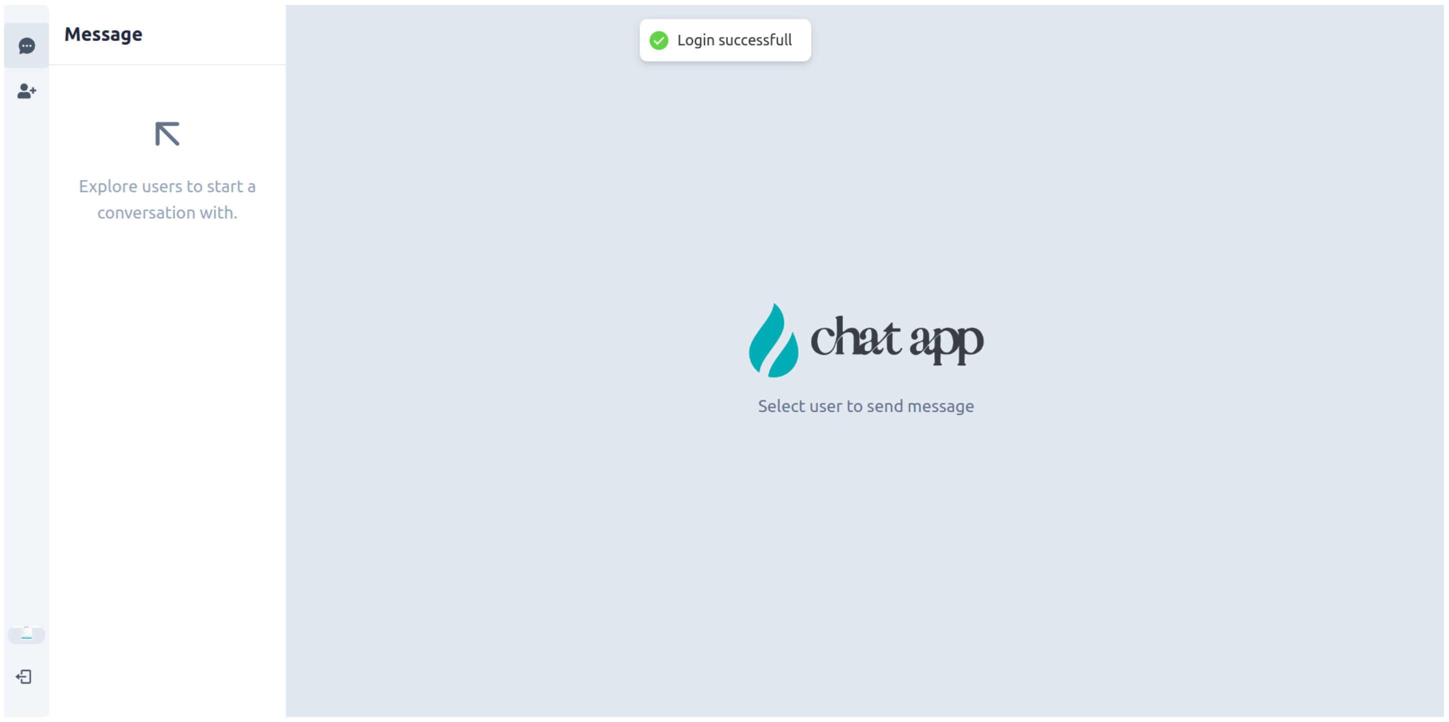
If the email is Incorrect:



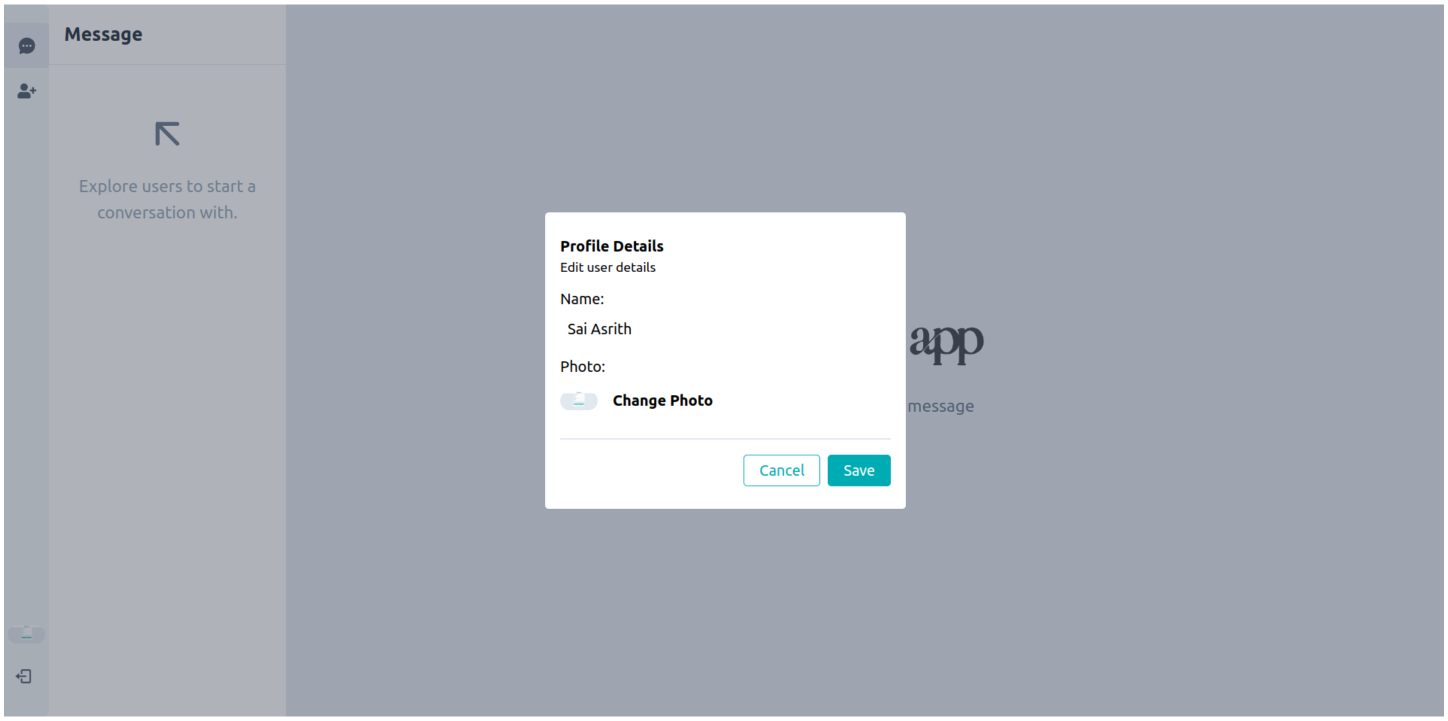
Home Page

After a successful login, the user is redirected to the **Home Page**. The Home Page includes several sections for the user to interact with:

- **Update User Details**: Users can update their personal information, including profile picture and display name.
- **Search Users**: Users can search for other registered users to start chatting.
- **Previous Chats**: A section to view and continue conversations with previously messaged users.
- **Log Out**: A logout button that allows users to securely log out of their account and return to the **Email Page**.


Message
Login successfull
Explore users to start a conversation with.
chat app
Select user to send message

User details and Updating user details


Message

Explore users to start a conversation with.

Profile Details

Edit user details

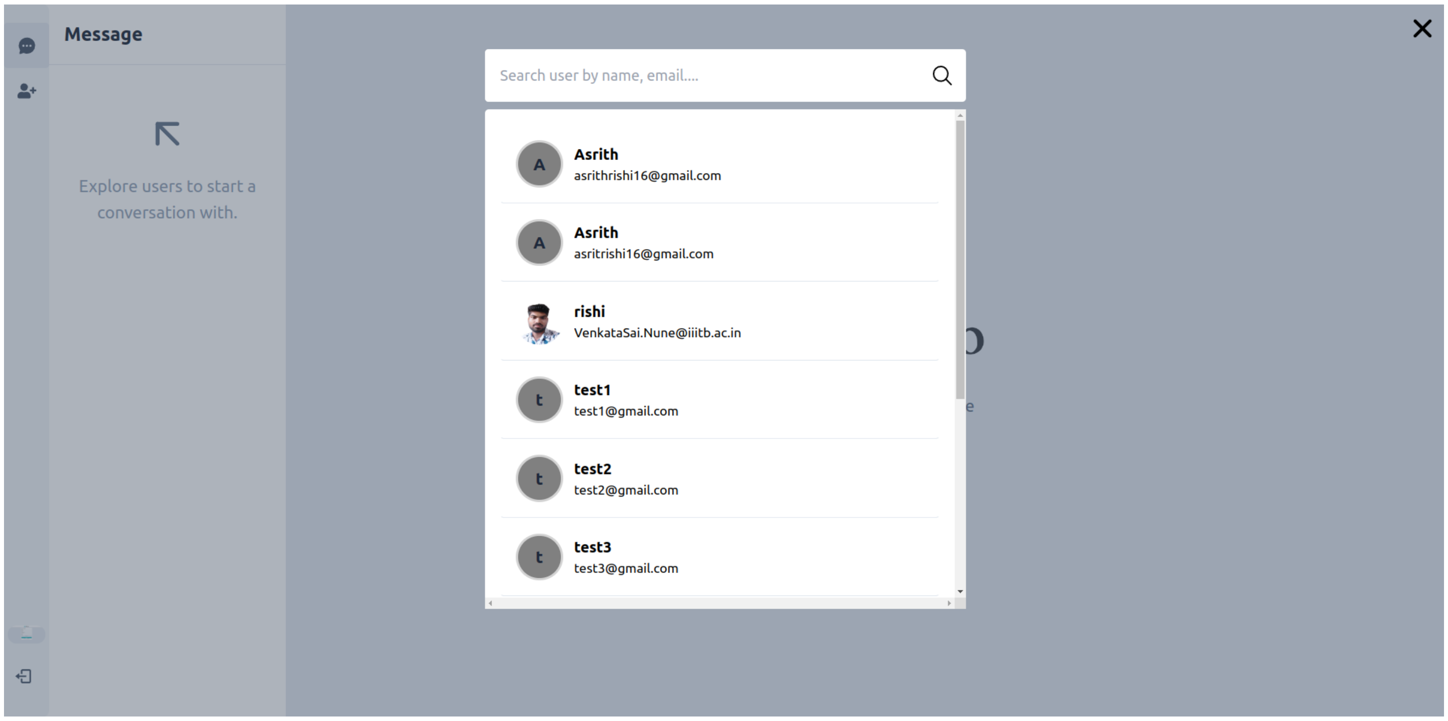
Name:

Sai Asrith

Photo:

Change Photo

To test if messaging works correctly, open a different browser and log in with another user account. Then, check if the messaging feature is functioning properly by sending messages between the two accounts. The search pages for both the users are attached below.


Message

Explore users to start a conversation with.

Search user by name, email....

Asrith
asrithrishi16@gmail.com

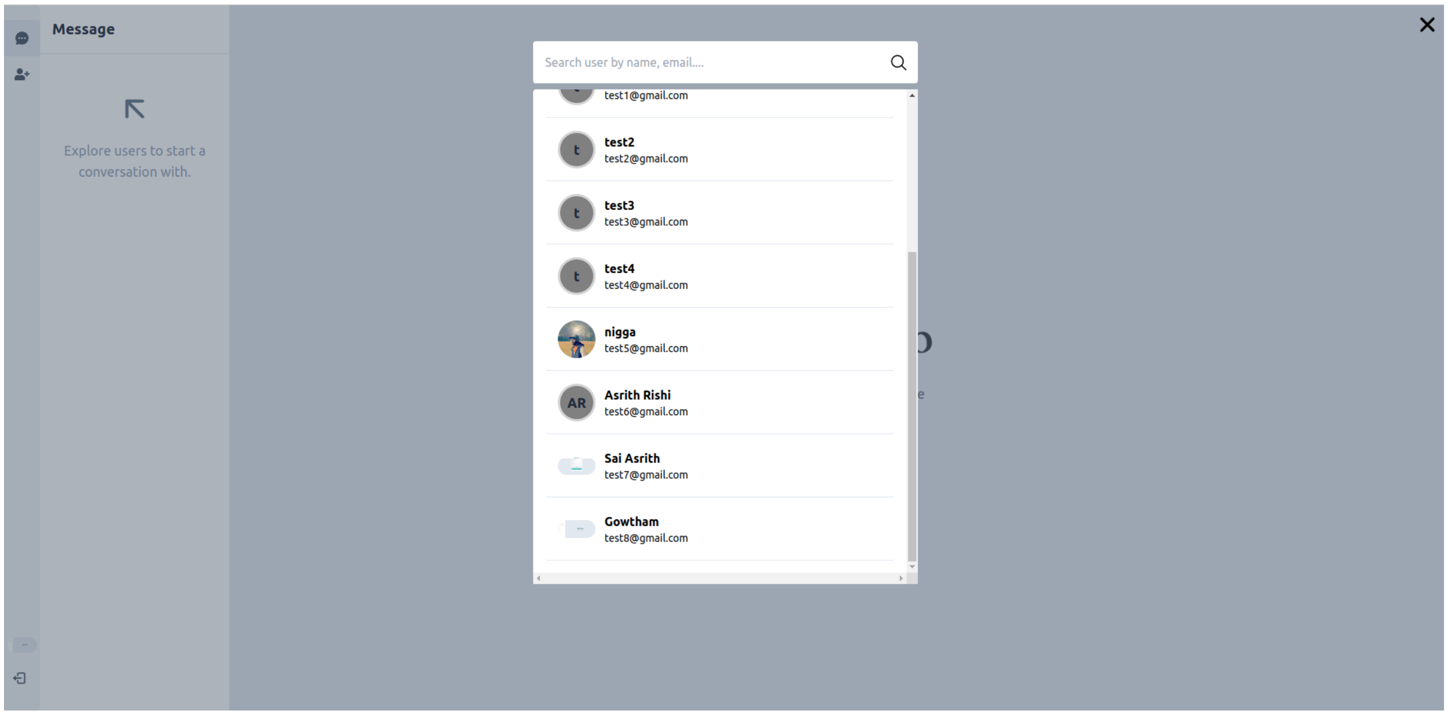
Asrith
asritrishi16@gmail.com

rishi
VenkataSai.Nune@iiitb.ac.in

test1
test1@gmail.com

test2
test2@gmail.com

test3
test3@gmail.com


Message
Explore users to start a conversation with.
Search user by name, email....

test1@gmail.com

test2
test2@gmail.com

test3
test3@gmail.com

test4
test4@gmail.com

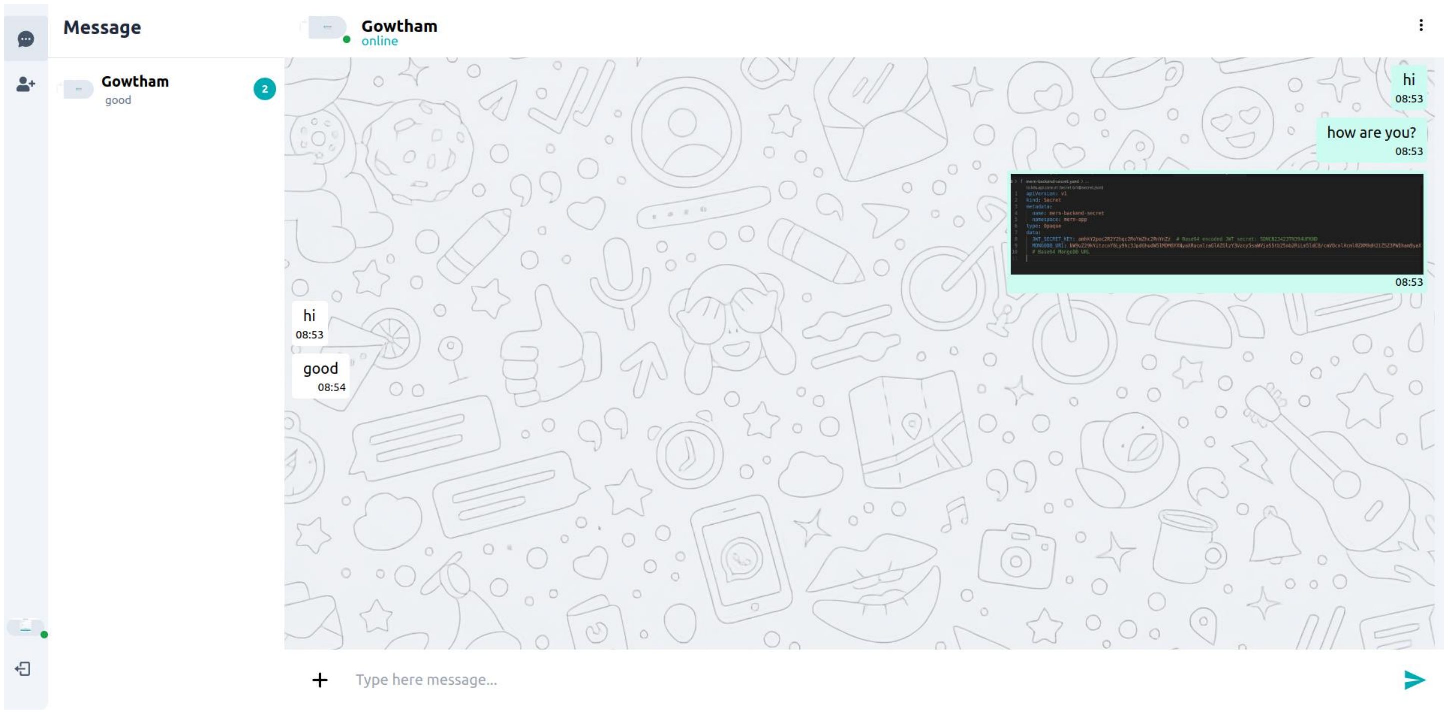
nigga
test5@gmail.com

Asrith Rishi
test6@gmail.com

Sai Asrith
test7@gmail.com

Gowtham
test8@gmail.com

Messaging pages of both users:



Gowtham
good
2

hi
08:53

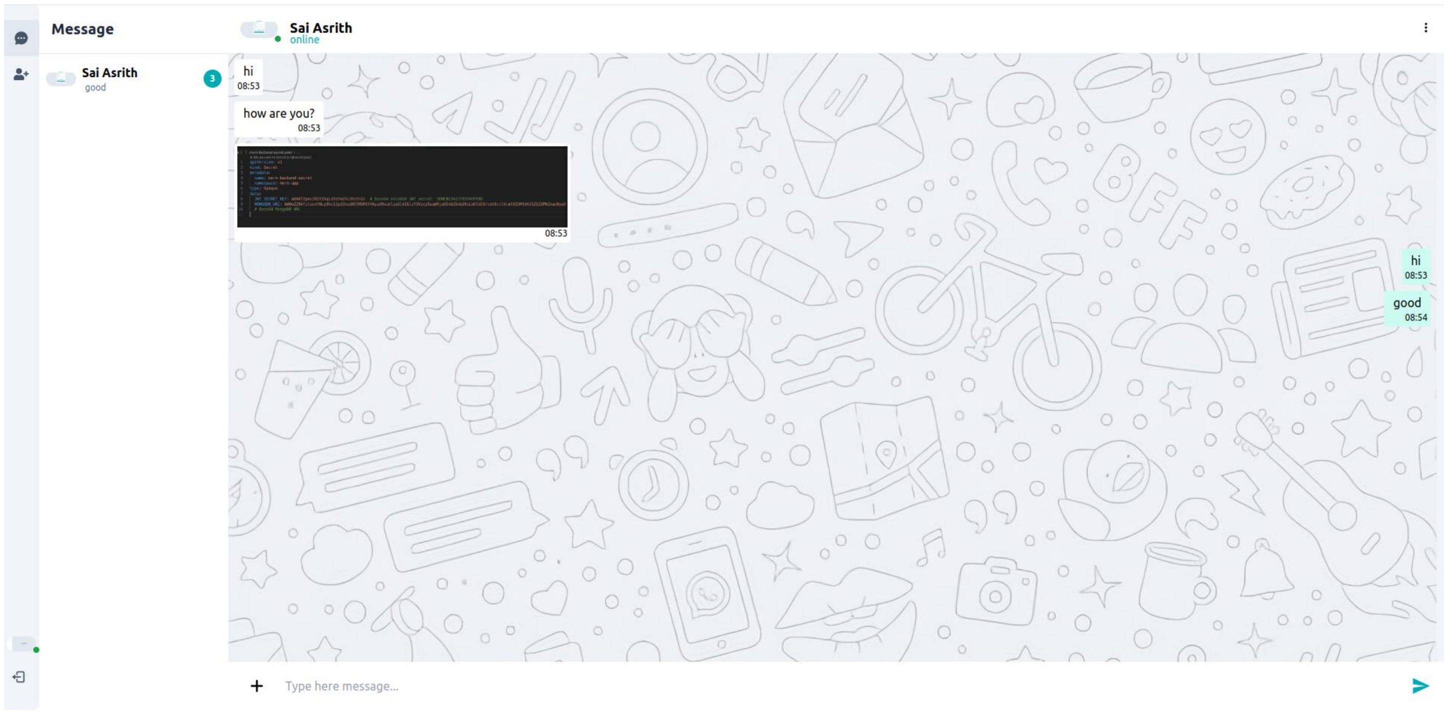
how are you?
08:53

```
mern-backend-secret.yaml > ...
apiVersion: v1
kind: Secret
metadata:
  name: mern-backend-secret
  namespace: mern-app
type: Opaque
data:
  JWT_SECRET_KEY: amhKY2poc2R2Y2hqc2RoYmZhc2RnYnZz  # Base64 encoded JWT secret: [illegible]
  MONGODB_URI: [illegible]
  # Base64 MongoDB URL
```

08:53

hi
08:53

good
08:54



hi
08:53

how are you?
08:53

```
apiVersion: v1
kind: Secret
metadata:
  name: mern-backend-secret
  namespace: mern-app
type: Opaque
data:
  JWT_SECRET_KEY: [illegible]  # base64 encoded JWT secret: [illegible]
  MONGODB_URL: [illegible]
  # Base64 MongoDB URL
```

08:53

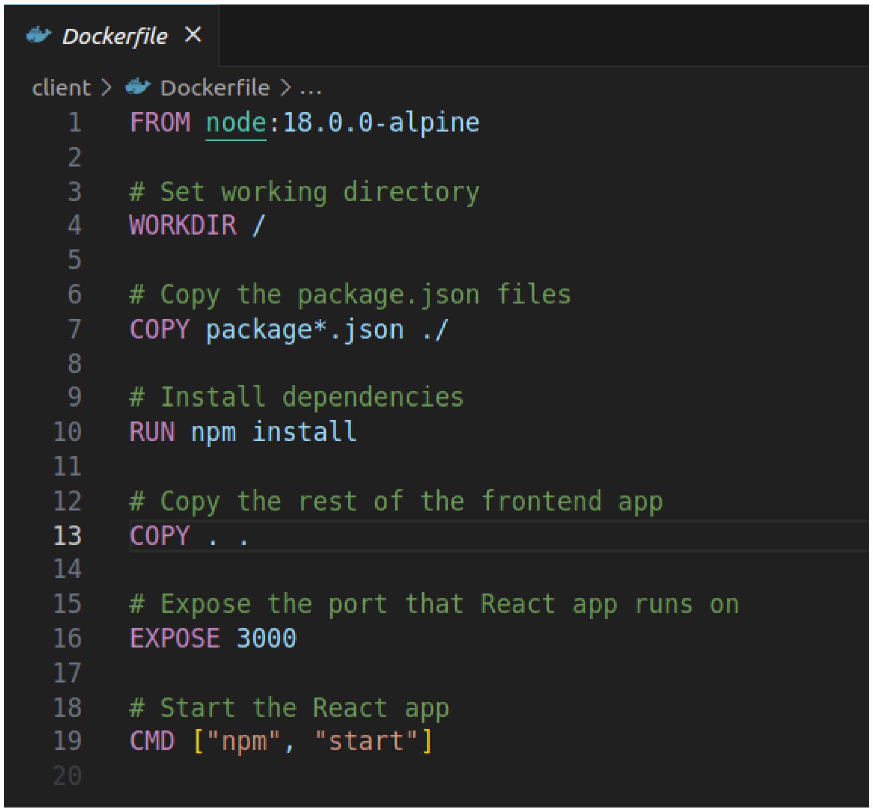
hi
08:53

good
08:54

From the pictures, we can see that the **Online** status option appears below the user names. If you want to log in with another user, simply log out and then log in with the new user's credentials. Additionally, you can update your user details, such as **name**, **profile picture**, and other information, directly from the **Home Page**.

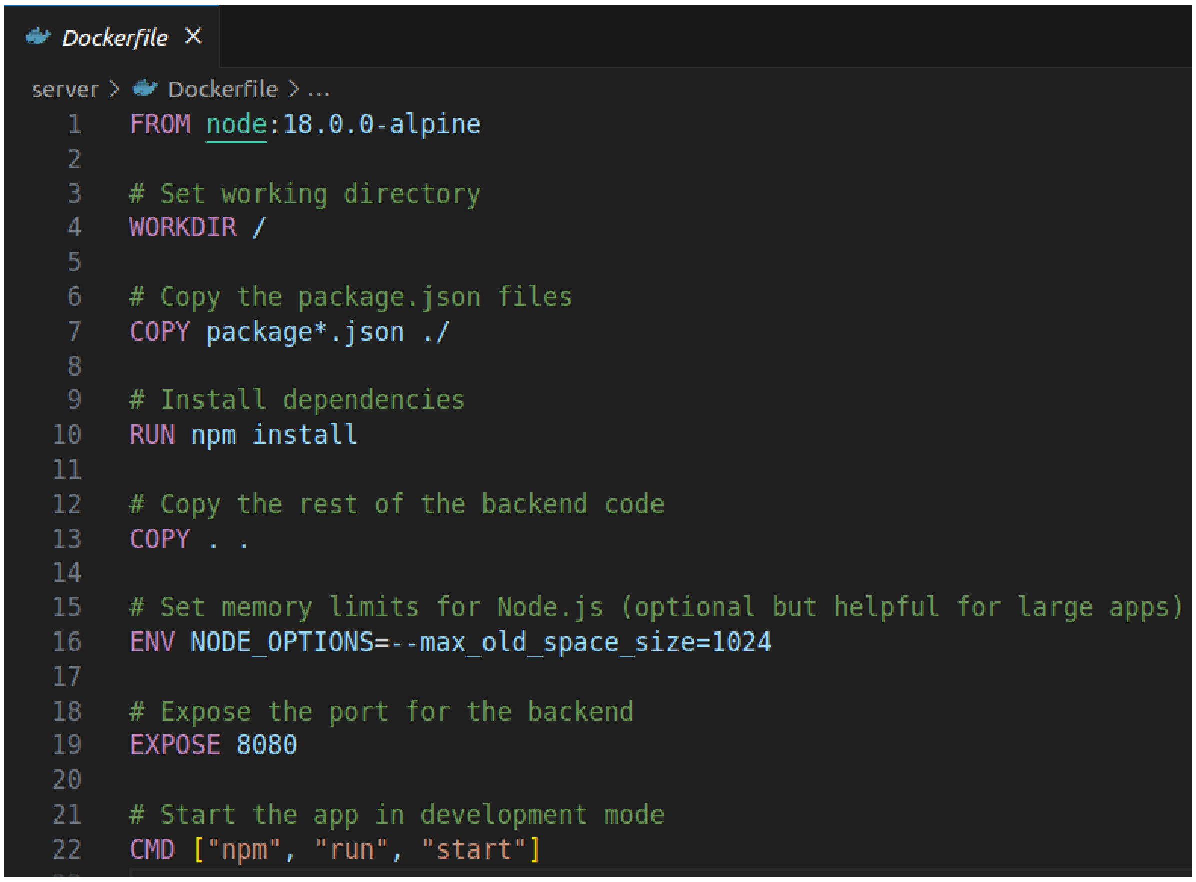
Docker

Frontend Docker File:



```
FROM node:18.0.0-alpine

# Set working directory
WORKDIR /

# Copy the package.json files
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the frontend app
COPY . .

# Expose the port that React app runs on
EXPOSE 3000

# Start the React app
CMD ["npm", "start"]
```

Backend Docker File:



```
FROM node:18.0.0-alpine

# Set working directory
WORKDIR /

# Copy the package.json files
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the backend code
COPY . .

# Set memory limits for Node.js (optional but helpful for large apps)
ENV NODE_OPTIONS=--max_old_space_size=1024

# Expose the port for the backend
EXPOSE 8080

# Start the app in development mode
CMD ["npm", "run", "start"]
```

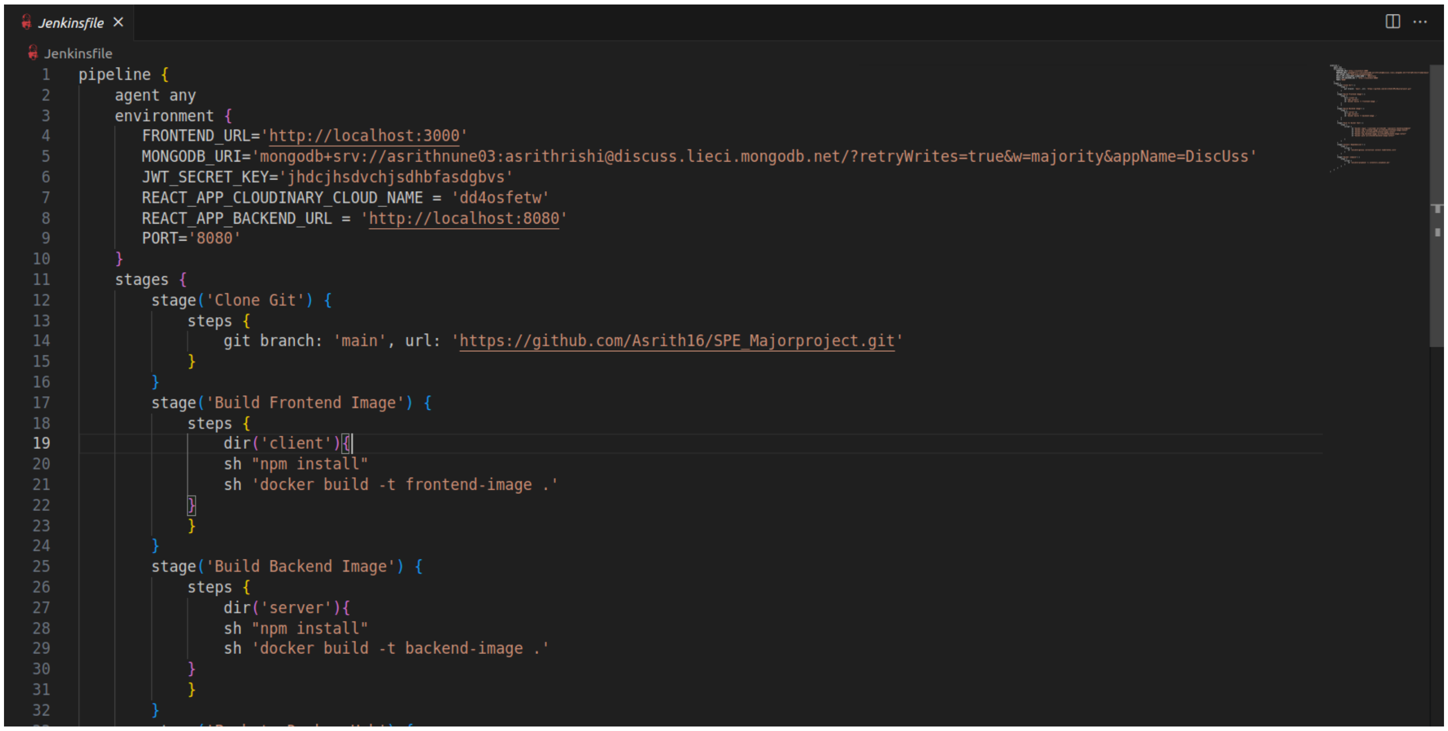
This **Dockerfile** is for a full-stack application with both backend and frontend components. It starts with the **Node.js 18.0.0-alpine** image as the base, sets the working directory to the root (/), and copies the package.json files into the container. It then installs the dependencies using npm install. The remaining backend code is copied into the container, and memory limits for Node.js are set to optimize performance. The Dockerfile exposes port **8080** for backend communication. Finally, it starts the application in development mode using npm run start.

Jenkins

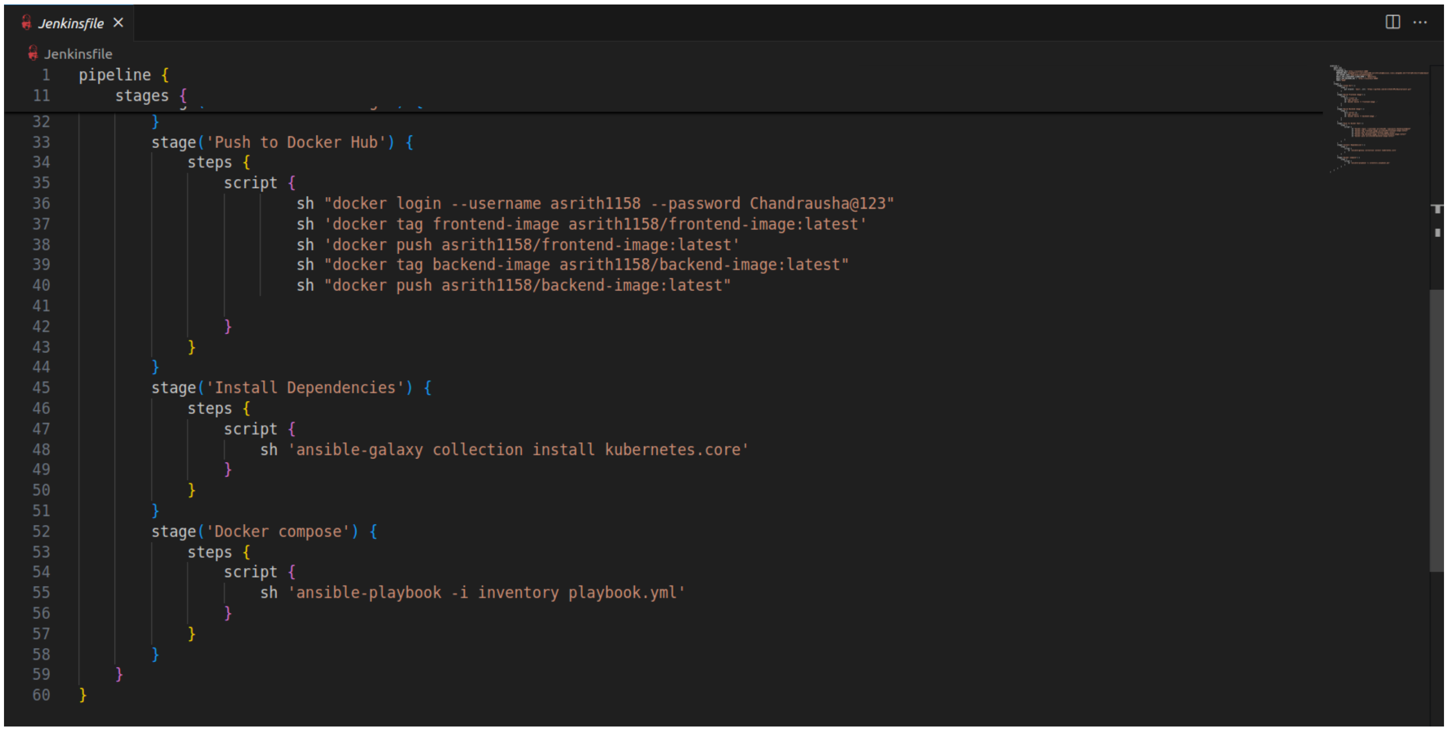
We used Jenkins pipeline scm from GitHub. The pipeline script was cloned from the GitHub repository And the code was also cloned from the same repository.

- **Stage 1: Clone Git**: Clones the project repository from GitHub using the main branch.
- **Stage 2: Build Frontend Image**: Installs frontend dependencies and builds a Docker image for the frontend.
- **Stage 3: Build Backend Image**: Installs backend dependencies and builds a Docker image for the backend.
- **Stage 4: Push to Docker Hub**: Logs in to Docker Hub and pushes the built frontend and backend images.
- **Stage 5: Install Dependencies**: Installs the required Ansible Kubernetes collection.
- **Stage 6: Docker Compose**: Runs an Ansible playbook to deploy the Docker containers.

We can use the docker [image/container] prune command to delete old docker and dangling docker images/containers.

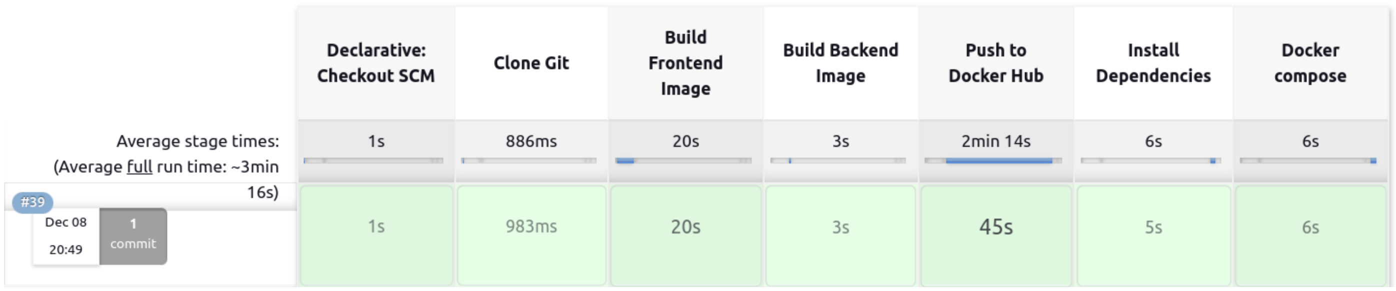


```
pipeline {
    agent any
    environment {
        FRONTEND_URL='http://localhost:3000'
        MONGODB_URI='mongodb+srv://asrithnune03:asrithrishi@discuss.lieci.mongodb.net/?retryWrites=true&w=majority&appName=DiscUss'
        JWT_SECRET_KEY='jhdcjhsdvchjsdhbfasdgbvs'
        REACT_APP_CLOUDINARY_CLOUD_NAME = 'dd4osfetw'
        REACT_APP_BACKEND_URL = 'http://localhost:8080'
        PORT='8080'
    }
    stages {
        stage('Clone Git') {
            steps {
                git branch: 'main', url: 'https://github.com/Asrith16/SPE_Majorproject.git'
            }
        }
        stage('Build Frontend Image') {
            steps {
                dir('client'){
                sh "npm install"
                sh 'docker build -t frontend-image .'
            }
            }
        }
        stage('Build Backend Image') {
            steps {
                dir('server'){
                sh "npm install"
                sh 'docker build -t backend-image .'
            }
            }
        }
```



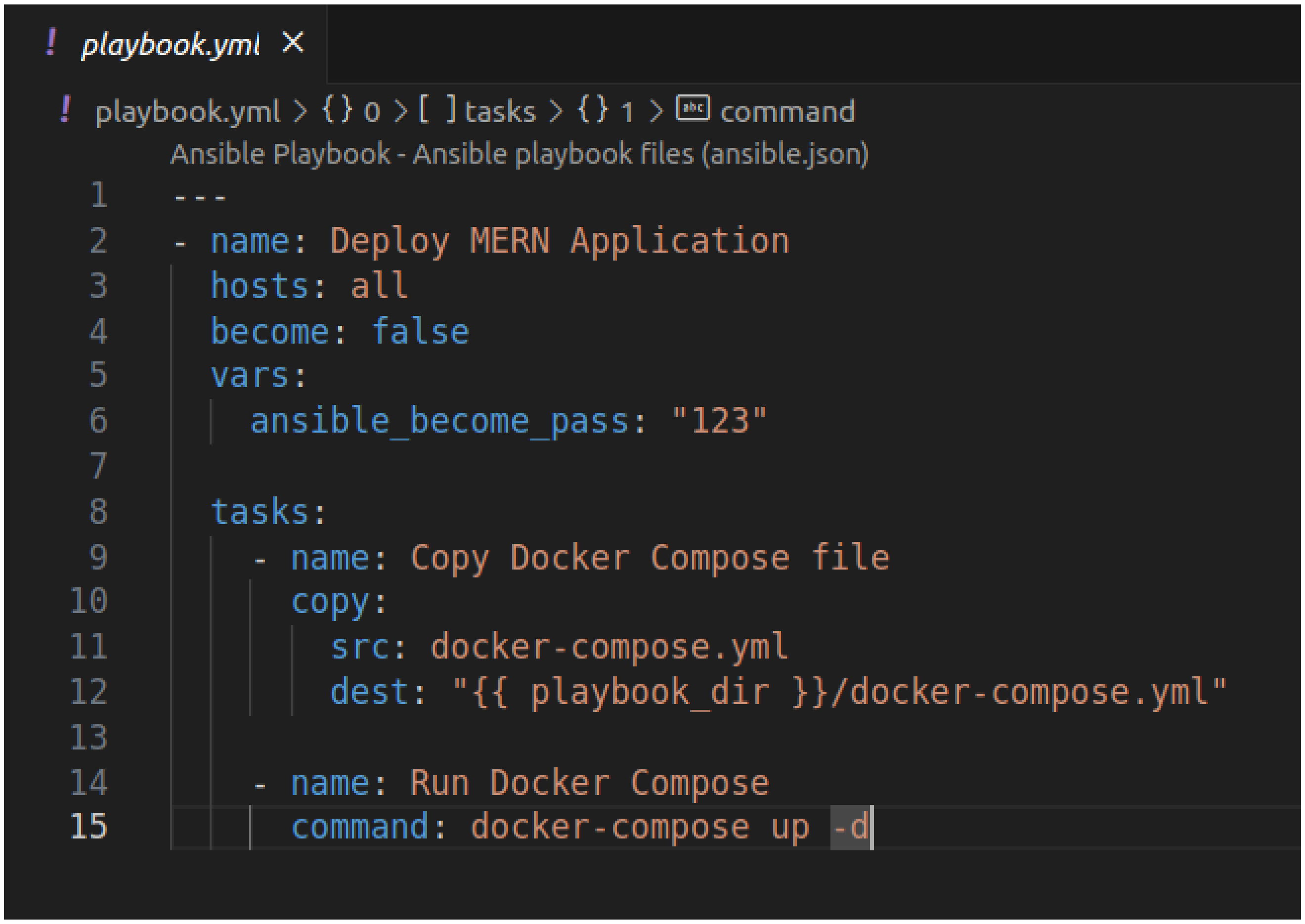
```
pipeline {
    stages {
        }
        stage('Push to Docker Hub') {
            steps {
                script {
                    sh "docker login --username asrith1158 --password Chandrausha@123"
                    sh 'docker tag frontend-image asrith1158/frontend-image:latest'
                    sh 'docker push asrith1158/frontend-image:latest'
                    sh "docker tag backend-image asrith1158/backend-image:latest"
                    sh "docker push asrith1158/backend-image:latest"

                }
            }
        }
        stage('Install Dependencies') {
            steps {
                script {
                    sh 'ansible-galaxy collection install kubernetes.core'
                }
            }
        }
        stage('Docker compose') {
            steps {
                script {
                    sh 'ansible-playbook -i inventory playbook.yml'
                }
            }
        }
    }
}
```

Jenkins Pipeline Result:



	Declarative: Checkout SCM	Clone Git	Build Frontend Image	Build Backend Image	Push to Docker Hub	Install Dependencies	Docker compose
Average stage times: (Average full run time: ~3min 16s)	1s	886ms	20s	3s	2min 14s	6s	6s
#39 Dec 08 20:49 1 commit	1s	983ms	20s	3s	45s	5s	6s

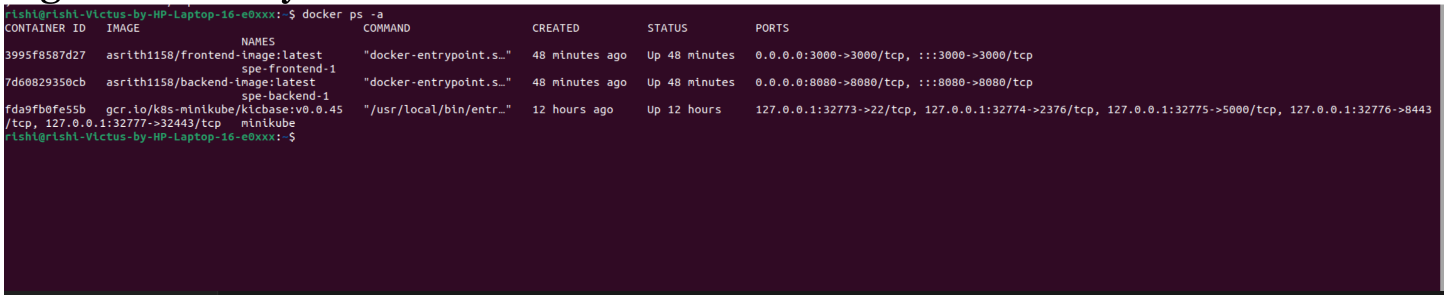
Playbook.yml has the permissions to the docker-compose and start docker images.


! playbook.yml ×
! playbook.yml > {} 0 > [ ] tasks > {} 1 > command
Ansible Playbook - Ansible playbook files (ansible.json)

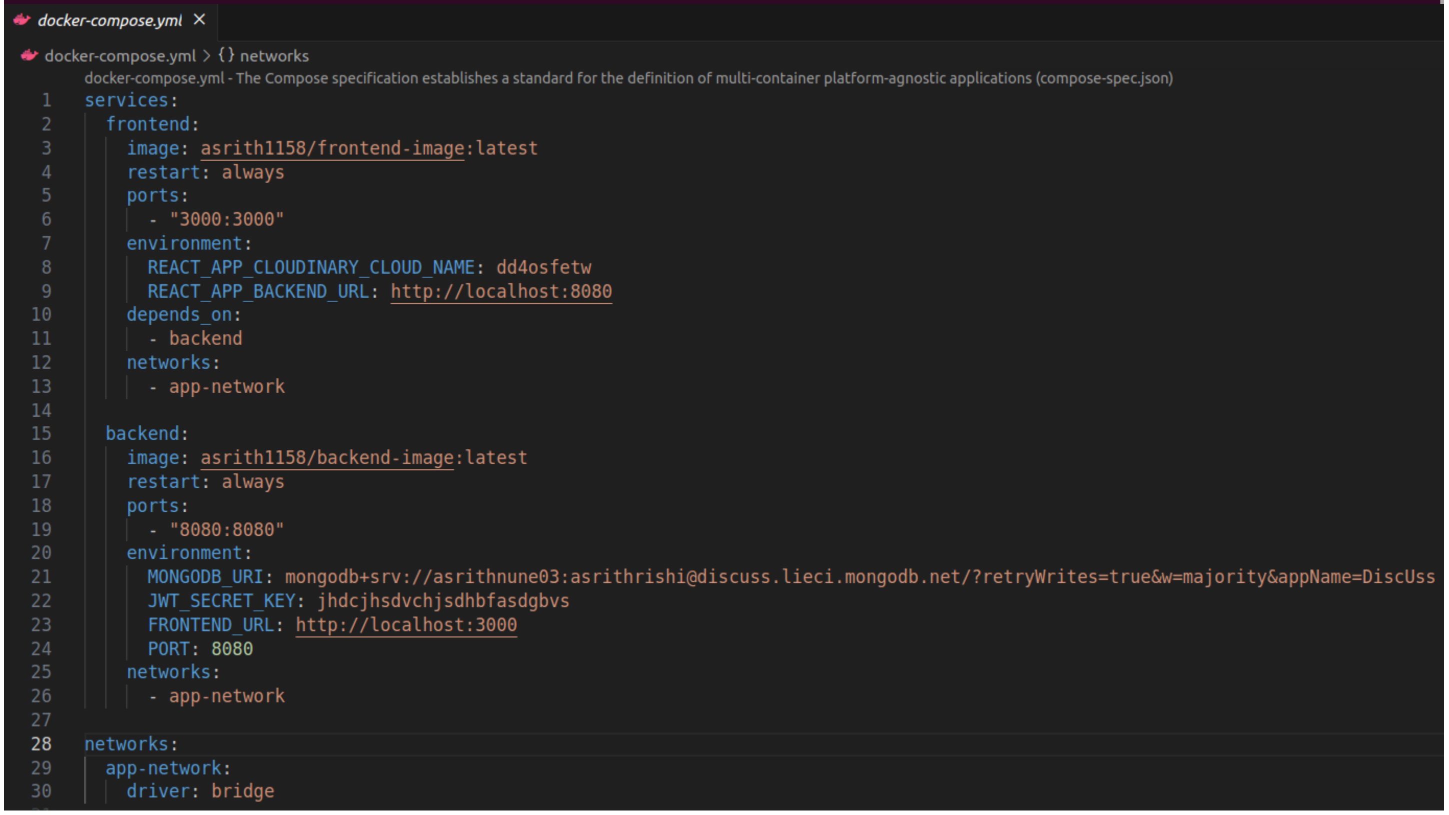
```yaml
---
- name: Deploy MERN Application
  hosts: all
  become: false
  vars:
    ansible_become_pass: "123"

  tasks:
    - name: Copy Docker Compose file
      copy:
        src: docker-compose.yml
        dest: "{{ playbook_dir }}/docker-compose.yml"

    - name: Run Docker Compose
      command: docker-compose up -d
```

Docker Compose

This **Docker Compose** file defines a multi-container environment for a full-stack application consisting of a **frontend** and **backend** service. It includes two services: **frontend** and **backend**. The **frontend** container uses the image asrith1158/frontend-image:latest, exposing port 3000, and is connected to the backend service through the app-network network. The **backend** container uses the image asrith1158/backend-image:latest, exposing port 8080, and relies on MongoDB connection details specified in the environment variables. The containers are set to restart always, ensuring high availability.



```
rishi@rishi-Victus-by-HP-Laptop-16-e0xxx:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS
                              NAMES
3995f8587d27   asrith1158/frontend-image:latest     "docker-entrypoint.s…"   48 minutes ago   Up 48 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
                              spe-frontend-1
7d60829350cb   asrith1158/backend-image:latest      "docker-entrypoint.s…"   48 minutes ago   Up 48 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
                              spe-backend-1
fda9fb0fe55b   gcr.io/k8s-minikube/kicbase:v0.0.45  "/usr/local/bin/entr…"   12 hours ago     Up 12 hours     127.0.0.1:32773->22/tcp, 127.0.0.1:32774->2376/tcp, 127.0.0.1:32775->5000/tcp, 127.0.0.1:32776->8443
/tcp, 127.0.0.1:32777->32443/tcp   minikube
rishi@rishi-Victus-by-HP-Laptop-16-e0xxx:~$
```



docker-compose.yml

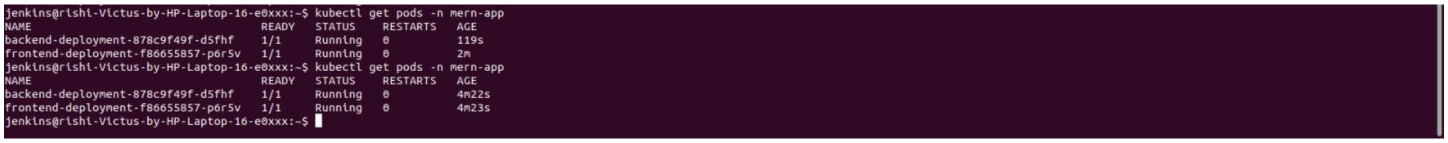
docker-compose.yml > {} networks

docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)

```yaml
services:
  frontend:
    image: asrith1158/frontend-image:latest
    restart: always
    ports:
      - "3000:3000"
    environment:
      REACT_APP_CLOUDINARY_CLOUD_NAME: dd4osfetw
      REACT_APP_BACKEND_URL: http://localhost:8080
    depends_on:
      - backend
    networks:
      - app-network

  backend:
    image: asrith1158/backend-image:latest
    restart: always
    ports:
      - "8080:8080"
    environment:
      MONGODB_URI: mongodb+srv://asrithnune03:asrithrishi@discuss.lieci.mongodb.net/?retryWrites=true&w=majority&appName=DiscUss
      JWT_SECRET_KEY: jhdcjhsdvchjsdhbfasdgbvs
      FRONTEND_URL: http://localhost:3000
      PORT: 8080
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

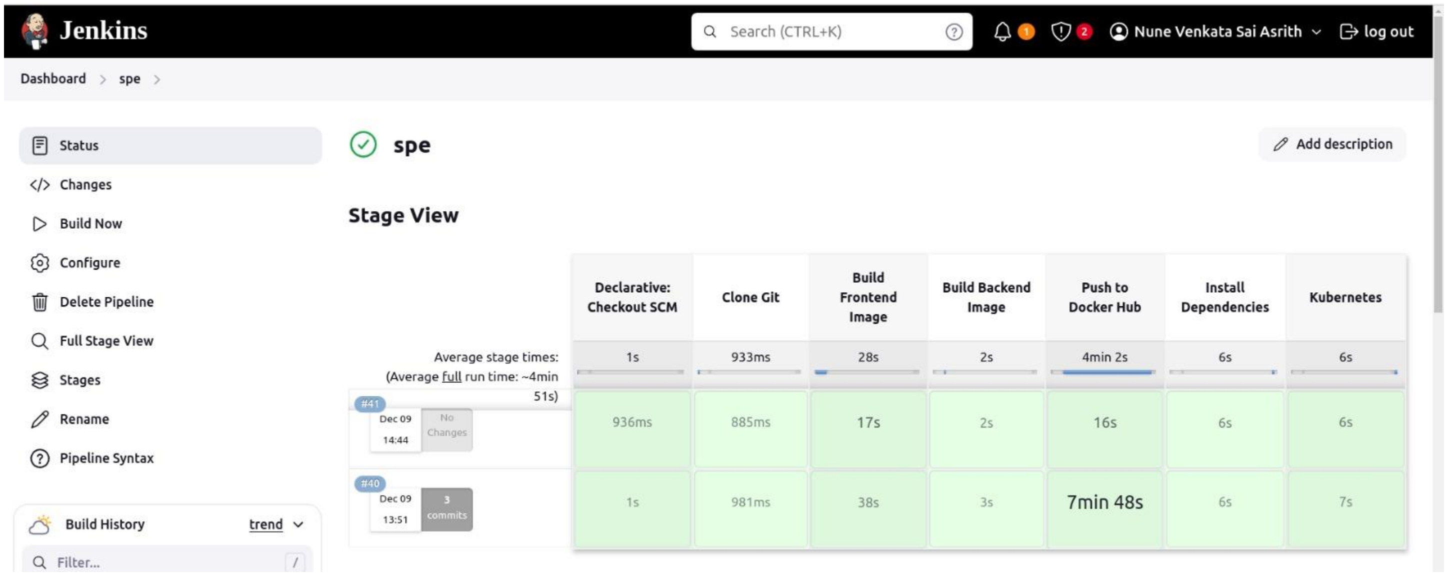
K8

We used Kubernetes to deploy the front-end, back-end containers in minikube environment, we have created a new namespace called mern-app and established connection through for the front-end pods and back-end pods using the addresses minikube-ip:nodePort.



```
jenkins@rishi-Victus-by-HP-Laptop-16-e0xxx:~$ kubectl get pods -n mern-app
NAME                                 READY   STATUS    RESTARTS   AGE
backend-deployment-878c9f49f-d5fhf   1/1     Running   0          119s
frontend-deployment-f86655857-p6r5v  1/1     Running   0          2m
jenkins@rishi-Victus-by-HP-Laptop-16-e0xxx:~$ kubectl get pods -n mern-app
NAME                                 READY   STATUS    RESTARTS   AGE
backend-deployment-878c9f49f-d5fhf   1/1     Running   0          4m22s
frontend-deployment-f86655857-p6r5v  1/1     Running   0          4m23s
jenkins@rishi-Victus-by-HP-Laptop-16-e0xxx:~$ 
```

The above pictures show the pods running in Kubernetes.



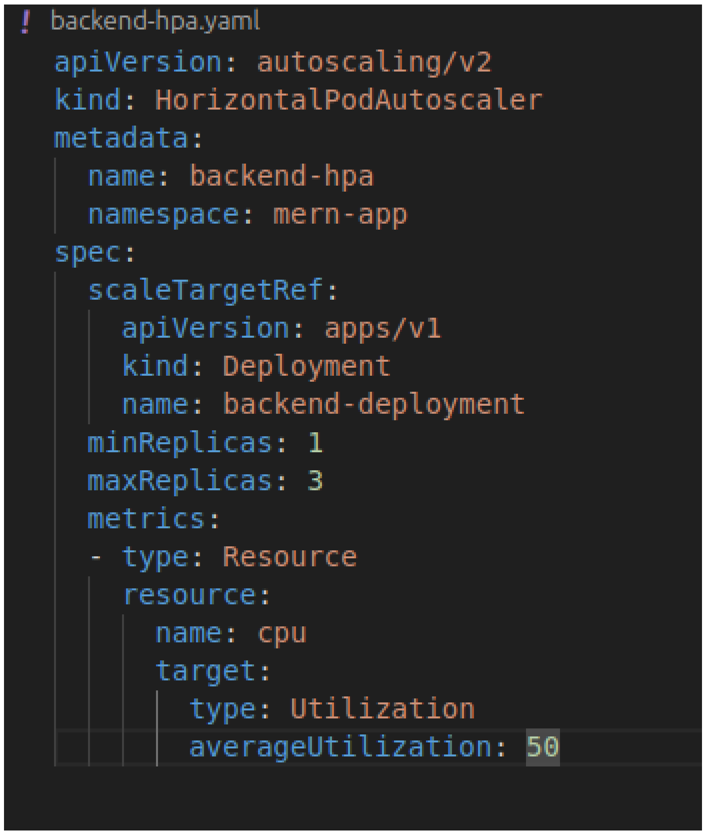
spe

Stage View

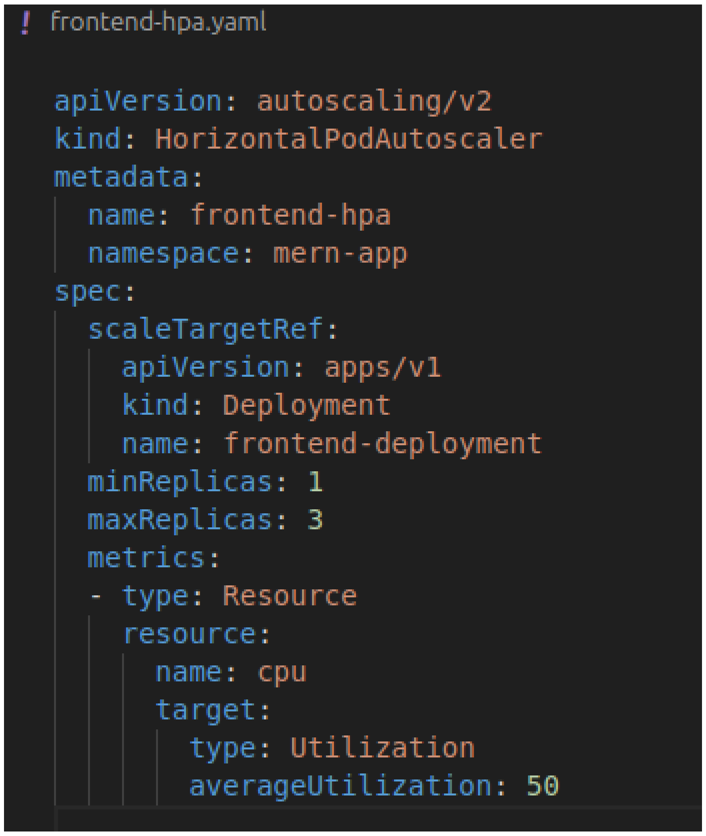
	Declarative: Checkout SCM	Clone Git	Build Frontend Image	Build Backend Image	Push to Docker Hub	Install Dependencies	Kubernetes
Average stage times: (Average full run time: ~4min 51s)	1s	933ms	28s	2s	4min 2s	6s	6s
#41 Dec 09 14:44 No Changes	936ms	885ms	17s	2s	16s	6s	6s
#40 Dec 09 13:51 3 commits	1s	981ms	38s	3s	7min 48s	6s	7s

The above picture shows the result of the Kubernetes build in Jenkins.

We have also used **<u>Horizontal Pod Autoscaler</u>** for both backend and frontend deployments.


backend-hpa.yaml

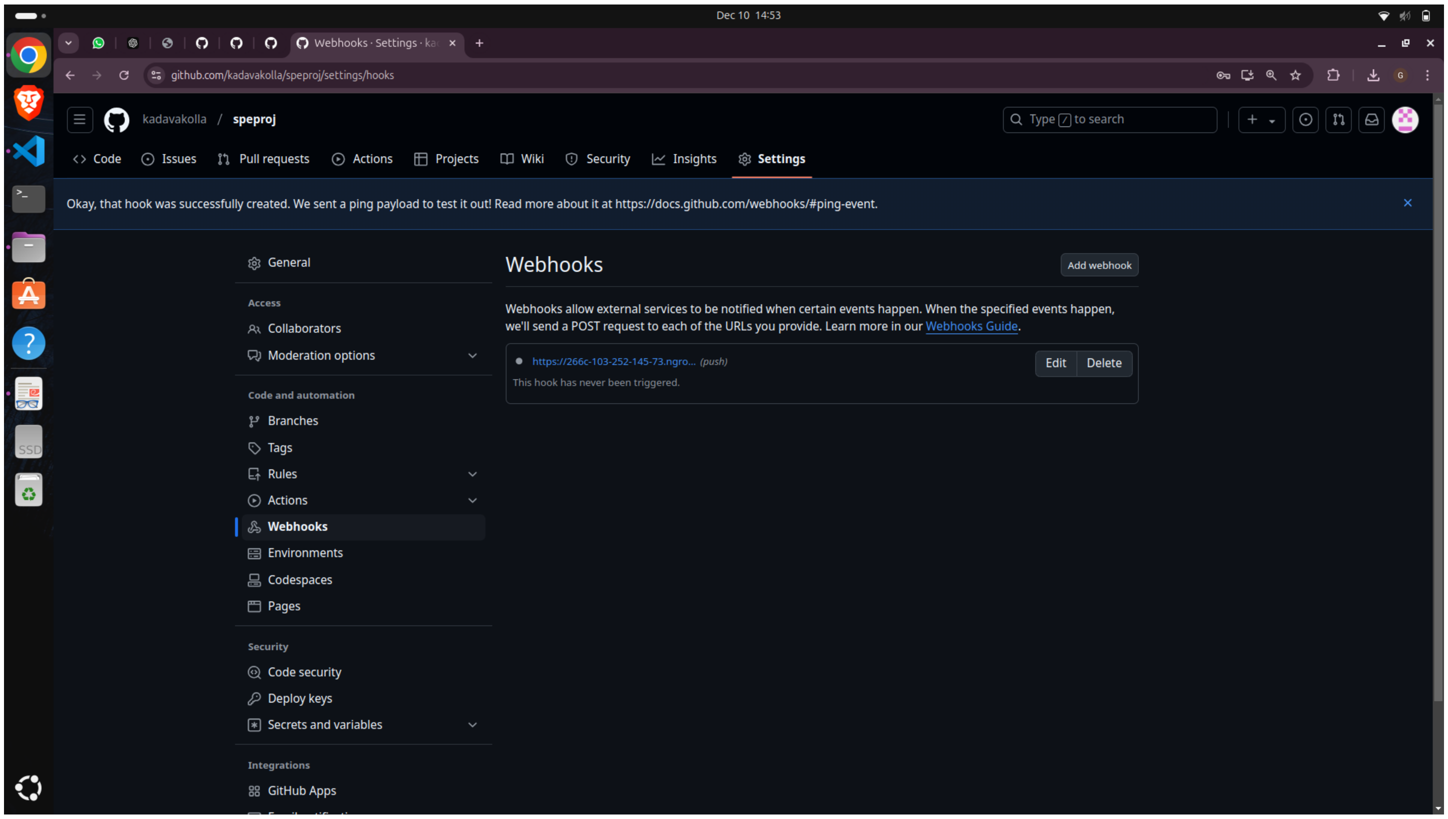
```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
  namespace: mern-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend-deployment
  minReplicas: 1
  maxReplicas: 3
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```


frontend-hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
  namespace: mern-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend-deployment
  minReplicas: 1
  maxReplicas: 3
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

Webhook:

We have also configured Jenkins' GitHub hook trigger for GITScm polling by creating a webhook for our project repository using the forwarding link of ngrok.



Okay, that hook was successfully created. We sent a ping payload to test it out! Read more about it at https://docs.github.com/webhooks/#ping-event.

Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

https://266c-103-252-145-73.ngro... (push)

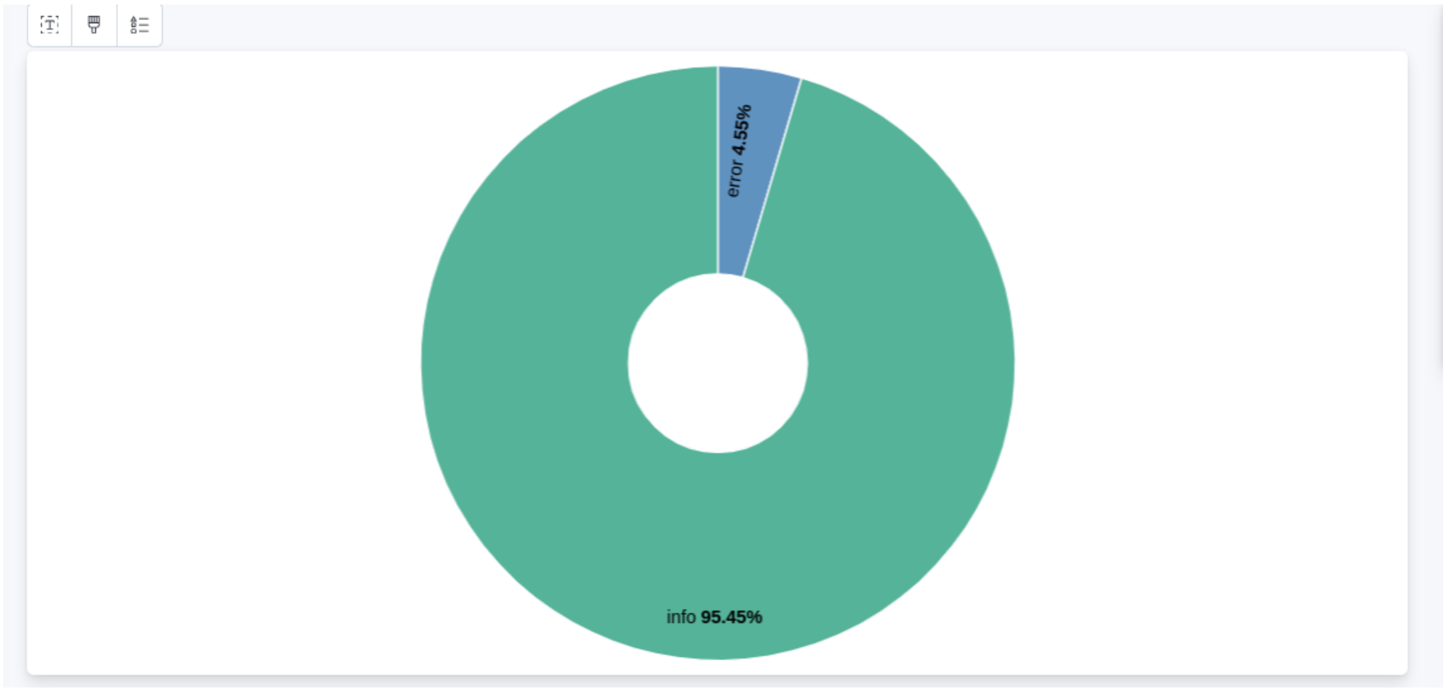
This hook has never been triggered.

ELK stack

Grok pattern(timestamp ("YYYY-MM-DD HH:mm:ss.SSS") + info/error + message))

ELK stack consists of 3 applications: Elasticsearch, Logstash, and Kibana. ELK stack is useful for collecting logs from all applications, analyzing these logs, and creating visualizations.

Each log has a Timestamp, deployment type (info/ error), and message.



Repository:

Github Repo: https://github.com/pavanreddy986/DiscUss-

Thank You