INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, BANGALORE

SOFTWARE PRODUCTION ENGINEERING - FINAL PROJECT

SOCIAL MEDIA WEBSITE

Stay connected with others

In the supervision of :- Prof B.Thangaraju
Teaching Assistant :- Vishal Rai
Team Member Details :-

Name	Roll No.
Prafull Pandey	MT2022150
Himanshu Digrase	MT2022155

20.04.2023

Table Of Contents

1. ABSTRACT

Our social media website is a platform where people can connect, share information, and engage with each other in meaningful ways. Users can create profiles, share updates, photos, and follow the activities of other users. Our website is designed to foster positive interactions and provide a safe, inclusive space for people of all backgrounds and identities. With powerful privacy and security features. Our social media website aims to bring people together and create a vibrant online community where individuals can express themselves, learn from others, and build meaningful relationships.

2. INTRODUCTION

a. OVERVIEW

A social media website is an online platform where people can connect, share information, and engage with each other in a variety of ways. Social media websites typically allow users to create a profile, post updates, photos, and videos, join groups and communities, and follow the activities of other users.

One of the primary features of social media websites is the ability to interact with other users through comments, likes, and shares. This creates a sense of community and encourages users to engage with each other in positive and meaningful ways.

Social media websites can be used for a wide range of purposes, from personal communication and entertainment to business marketing and professional networking. They can also be used to share news and information, discuss social issues, and mobilize for social causes.

b. FEATURES

MERN app in which users can register/ sign in with image upload functionality and can like/ dislike any post of other users.

1. Users can login with the provided credentials.
2. Users can perform CRUD operations on posts with image upload functionality.
3. User can view their previous/present posts in news feed.
4. User can like/ dislike any post and can comment on any post.
5. User can add/ remove friend in their friend list.
6. User can view other user's profile and their recent activity.

c. WHY DevOps?

DevOps is a set of tools that automate software development and IT operations. It focuses on shortening the systems development life cycle and providing continuous delivery with high software quality. DevOps is one up on Agile software development; multiple DevOps aspects came from the Agile methodology.

1. It optimizes the overall business by increasing efficiency through automation.
2. Improves software development and deployment speed and stability.
3. Deployment failures, rollbacks, and recovery time are all reduced.
4. Improved collaboration and communication; lower costs and IT headcount

3. TOOLS USED

1. Version Control System : Git and GitHub
   https://github.com/himanshudigrase/SocialWeb
2. Continuous Integration/Continuous Delivery : Github Actions
3. Building Tool : NPM (Node Package Manager)
4. Testing : Mocha framework
5. Containerization : Docker
   https://hub.docker.com/r/hims0301/backend
   https://hub.docker.com/r/hims0301/frontend
6. Deployment : Ansible
7. Log creation, monitoring and visualization : ELK (Elasticsearch,Logstash, Kibana)
8. Frontend Development : React.js, Vite
9. Backend Development : Node.js, Express.js
10. Database : MongoDB

4. SYSTEM CONFIGRATION

1. Operating System - Linux Ubuntu 22
2. CPU & RAM - 4 core processor and 8GB RAM
3. Kernel Version - 5.4.0-89-generic
4. Database - MongoDB 4.4.5

5. SOFTWARE DEVELOPEMENT LIFECYCLE- SDLC

i. INSTALLATIONS

The frontend(client) of the project uses React.js and the backend(server) is built using Node.js, Express.js and the database is using MongoDB. To begin with, install nodejs on the system.

ii. NODEJS INSTALLATION

Node.js is a javascript programming package management. It is the JavaScript runtime environment Node.js' default package manager. It comes pre-installed with Node.js. The package.json file contains the definitions for all npm packages. Package.json's content must be in JSON format. The definition file must have at least two fields. The names and variants are as follows. It is capable of managing dependencies. It installs all of the project's dependencies in a single command line. The package.json file also defines dependencies.

Inorder to run React, node environment shall be installed before starting React app

```
sudo apt-get install nodejs
sudo apt-get install npm
```

Run npm install command inside the frontend and backend folder to install
all the necessary dependencies of the project.

```
sudo npm install
```

```
himanshu@pop-os:/media/himanshu/New Volume/SocialMediaWebsite/SocialFrontEnd$ cd socialfrontend/
himanshu@pop-os:/media/himanshu/New Volume/SocialMediaWebsite/SocialFrontEnd/socialfrontend$ npm install

  changed 12 packages, and audited 167 packages in 3s
```

Figure: Frontend npm install

```
himanshu@pop-os:/media/himanshu/New Volume/SocialMediaWebsite$ cd SocialBackend/
himanshu@pop-os:/media/himanshu/New Volume/SocialMediaWebsite/SocialBackend$ npm install

  up to date, audited 872 packages in 9s
```

Figure: Backend npm install

```
{} package.json > ...
You, last week | 1 author (You)
1  {
2    "type": "module",
3    "dependencies": {
4      "bcrypt": "^5.1.0",
5      "body-parser": "^1.20.2",
6      "cors": "^2.8.5",
7      "dotenv": "^16.0.3",
8      "express": "^4.18.2",
9      "gridfs-stream": "^1.1.1",
10     "helmet": "^6.0.1",
11     "jsonwebtoken": "^9.0.0",
12     "mongoose": "^7.0.3",
13     "morgan": "^1.10.0",
14     "multer": "^1.3.0",
15     "multer-gridfs-storage": "^5.0.2"
16   }
17 }
```

Figure: Backend package.json

FrontEnd > socialfrontend > {} package.json > {} dependencies > yup

```
{
  "name": "socialfrontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  ▷ Debug
  "scripts": {
    "dev": "vite",
    "start": "run dev --host",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.10.6",
    "@emotion/styled": "^11.10.6",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.12.0",
    "@reduxjs/toolkit": "^1.9.3",
    "dotenv": "^16.0.3",
    "formik": "^2.2.9",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-dropzone": "^14.2.3",
    "react-redux": "^8.0.5",
    "react-router-dom": "^6.10.0",
    "redux-persist": "^6.0.0",
    "yup": "^1.1.1"      Prafull Pandey, 4 weeks ago • state pe
  },
  "devDependencies": {
    "@types/react": "^18.0.28",
    "@types/react-dom": "^18.0.11",
    "@vitejs/plugin-react": "^3.1.0",
    "vite": "^4.2.0"
  }
}
```

Figure: Frontend package.json

iii. MONGODB

We create a mongodb database on atlas to store all the user and posts information. To set this database with our project use create URI and store it in the .env file

```
.env
MONGO_URL = 'mongodb+srv://<username>:<password>@socialmediacluster.1vyzwup.mongodb.net/?retryWrites=true&w=majority'
PORT = 3001
```

Figure: MONGO_URL

iv. Create a cluster on MONGO-DB Atlas

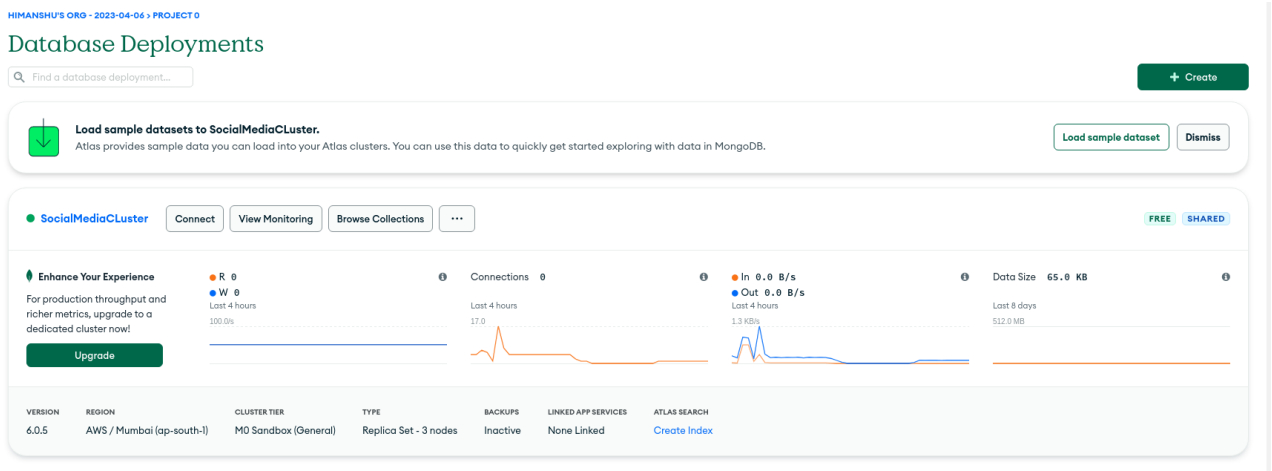

Figure: MONGO Database

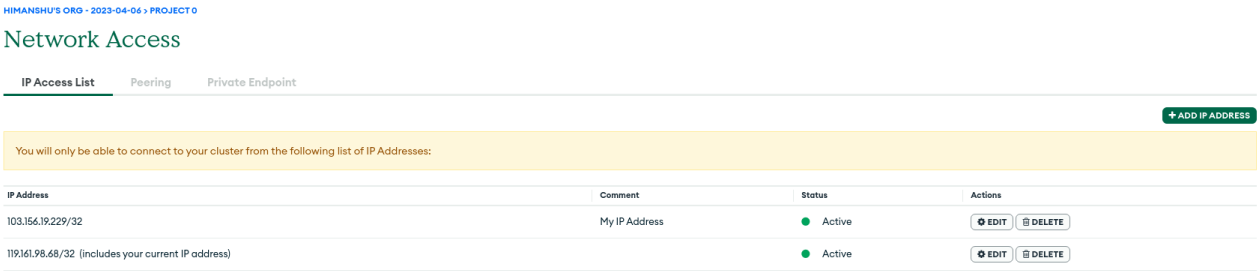

Figure: MONGO Network

Set Network Access to 0.0.0.0/0 so that the website can be accessed from all IP addresses. This is ideal for real time development project, for development we can set a few specific addresses.

v. Connect app to database

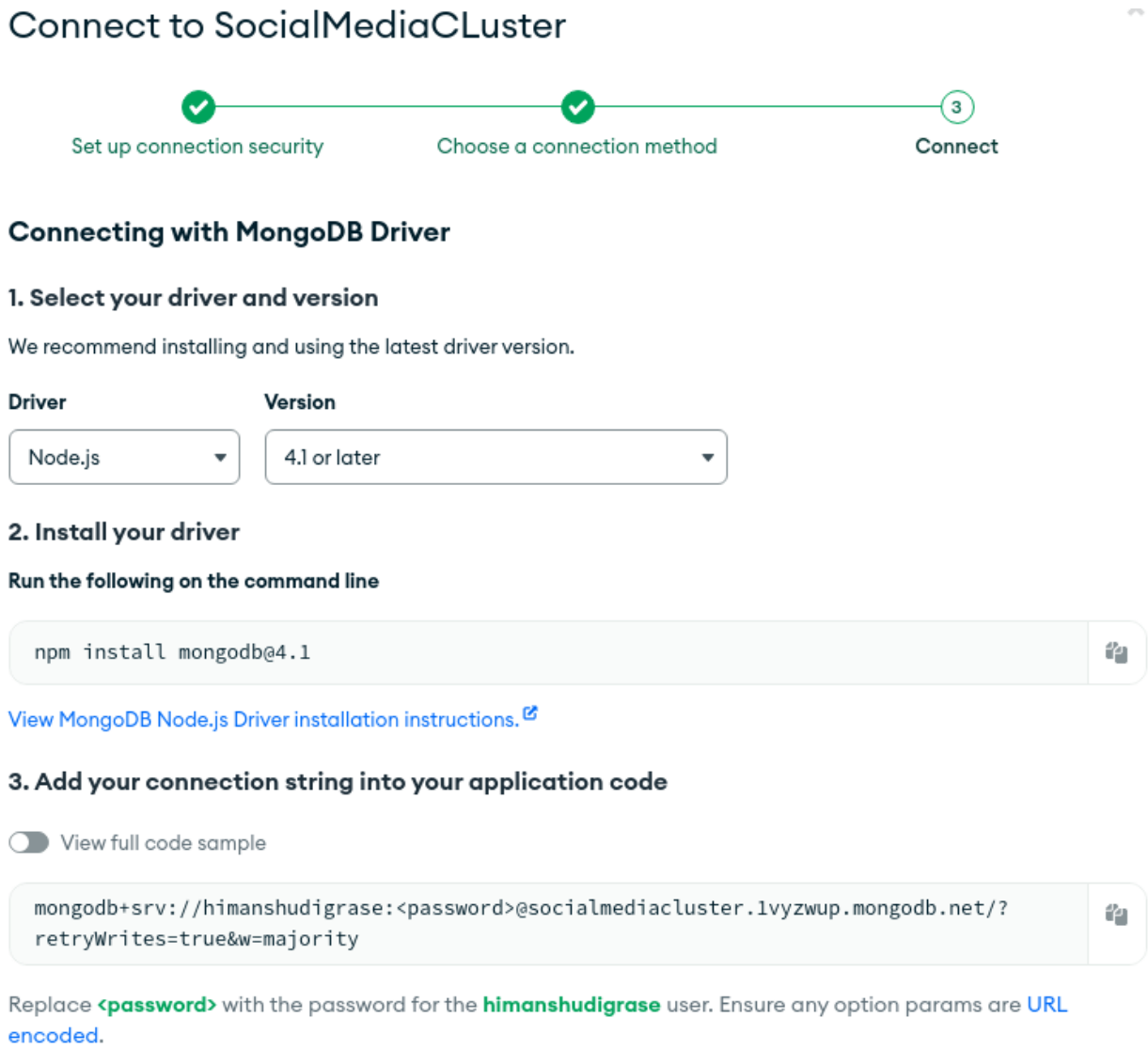

Figure: Connection string for moongodb

Add the above URL to .env to connect to the database. Replace the username with mongodb's username and set the password for the same. Change the database name at myFirstDatabase, to your specific database name.

Database Entries from - Browse Collections

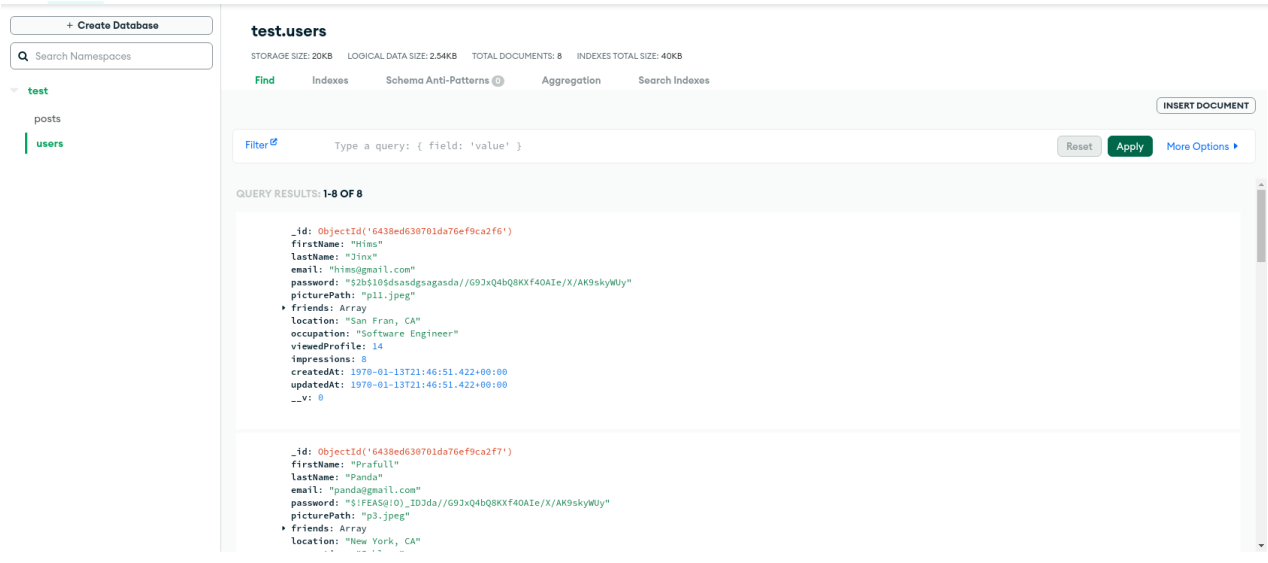

Figure: Database Collection

vi. Store MongoDB URL in .env file

By creating a.env file in the api (backend) subdirectory, you can save the cluster's URL. In that URL, provide all of the MongoDB Atlas' credentials (password). This.env file is part of.gitignore. Furthermore, Mongoose uses the variable used to store this URL to connect to MongoDB.

```
mongoose.connect(process.env.MONGO_URL,{
    useNewUrlParser: true,
    useUnifiedTopology: true,
})
.then(()=>{
    app.listen(PORT,()=>console.log(`Server Port: ${PORT}`));

}).catch((e)=>console.log(`${e}`));
```

Figure: Mongo Connect URL

vii. JWT Authentication

JSON Web Tokens (JWT) are an RFC 7519 open industry standard for representing claims between two parties. For example, you can use jwt.io to decode, verify, and produce JWT.

JWT defines a concise and self-contained way for transmitting information between two parties as a JSON object. This information may be reviewed and trusted because it is signed. A secret (using the HMAC algorithm) or an RSA or ECDSA public/private key pair can be used to sign JWTs.

We create the JWT token and add it to .env file for performing authentication.

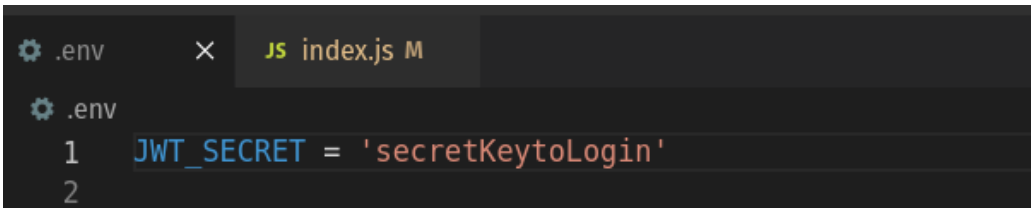

Figure: JWT Token

viii. Source Code Management

Source code management (SCM) is used to keep a track of all the modifications done to a source code repository. SCM tracks a current history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is very similar to Version control.

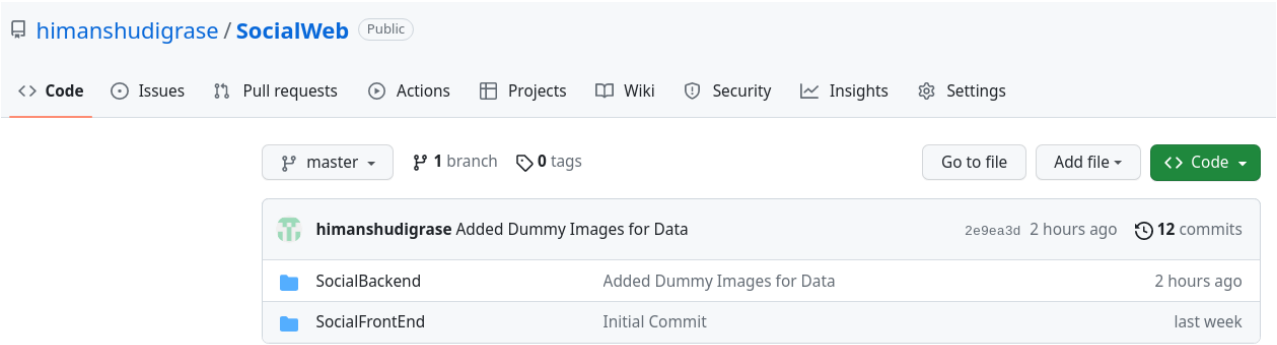

Figure: GITHUB REPO

To integrate project with GitHub we do the following steps :

1. git init - Initializes the project as a github repository locally.
2. git remote add origin - Add the details of the remote branch.
3. git add - This command stages all the changes of local repo.
4. git commit - Commit command commits the changes to the respective branch in the current repository.
5. git push - Push command will push the changes to the remote repository on GitHub.

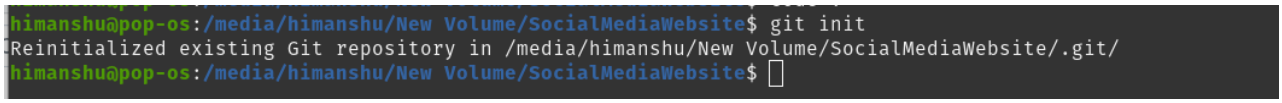

Figure: git init

ix. Testing

To make sure the project is running alright we provide a set of test cases. We use a library called 'Jest'. Jest is a JavaScript test framework running on Node. js and in the browser. Jest is a testing framework developed by Facebook. Originally designed to make UI testing easier for React developers, it's now a full standalone suite of tools for any type of JavaScript project (including Node.js) and includes features such as a built-in assertion library, code coverage, and mocking. Jest also

runs multiple test suites concurrently, which can speed up the overall testing process. The downside of parallel execution is it can make debugging tests more difficult.

ix. Testing

For anyone coming from a BDD-style of Mocha, Jest tests are pretty familiar looking. Jest adds several global functions to help with setting up and running tests, such as describe, it, expect, and the jest object (used mostly for mocking).
Backend Testing SS
Backend Testing SS2

x. Containerization

Docker is an open source platform for developing, shipping, and running applications. Docker enables users to separate your applications from your infrastructure so you can deliver software quickly. This allows us to deploy products directly to users' computers without installing each software one-by-one.

We need to create an account on DockerHub, which is a public registry.

We then push our created image on this repository, this is publicly available and can be pulled by any user and deployed on a local machine.

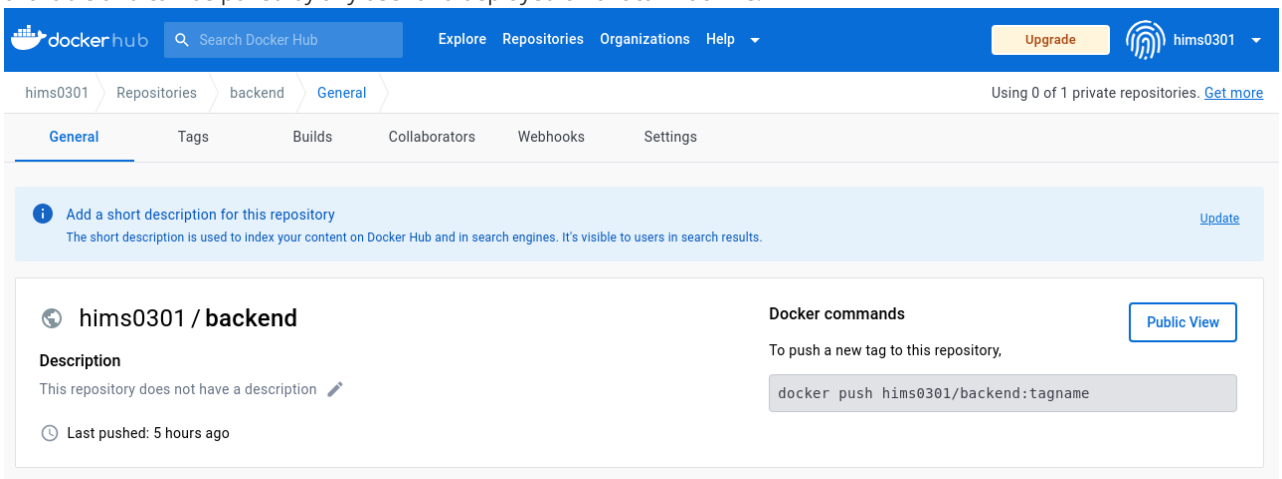

Figure: Dockerhub Backend Image

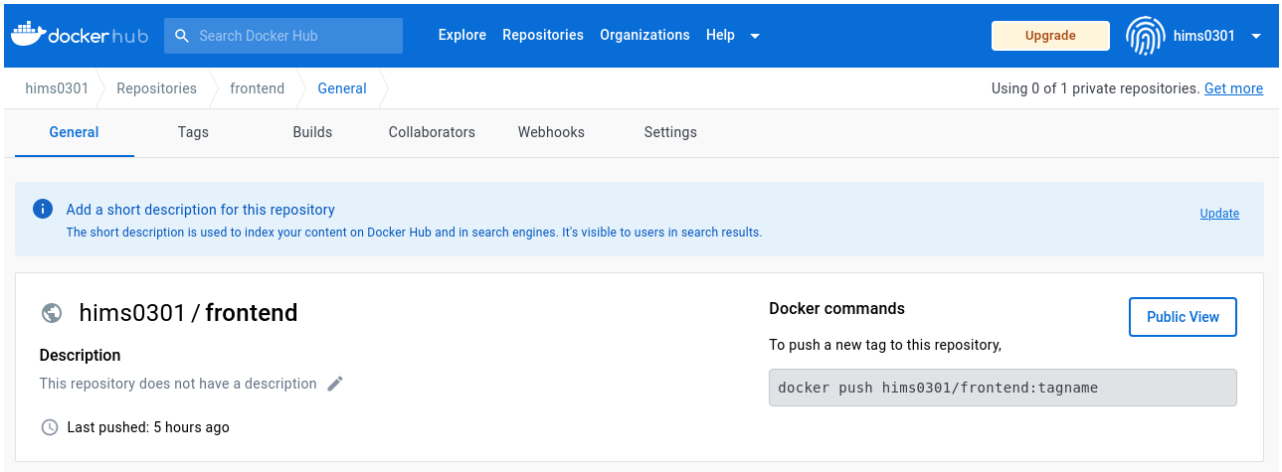

Figure: Dockerhub Frontend Image

xi. Create repository on docker-hub

Link :

a. https://hub.docker.com/r/hims0301/backend

b. https://hub.docker.com/r/hims0301/frontend

c. https://hub.docker.com/repositories

SocialBackend > Dockerfile > ...

You, 2 weeks ago | 1 author (You)

```
FROM node:17-alpine
ARG MONGO_URL
ARG PORT
ARG JWT_SECRET
RUN npm install -g nodemon

WORKDIR /app
COPY package.json .
RUN npm install

COPY . .
ENV MONGO_URL=$MONGO_URL
ENV PORT=$PORT
ENV JWT_SECRET=$JWT_SECRET
EXPOSE 3001

CMD ["nodemon", "index.js"]
```

Figure: Dockerfile Backend

SocialFrontEnd > socialfrontend > Dockerfile > ...

You, 1 second ago | 1 author (You)

```
FROM node:17-alpine

ARG PORT
WORKDIR /app
COPY package.json .

RUN apk add --no-cache xdg-utils curl && \
    npm install

COPY . .
ENV PORT=$PORT

EXPOSE 6002
CMD ["npm", "run", "dev"]
```

Figure: Dockerfile Frontend

Docker holds a set of all the commands a user could call on the command line to assemble an image. For our node project we set to copy the
package.json and package-lock.json to the container and then run npm install to install all the packages there.

xii. CI/CD Pipeline

Continuous Integration : Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into
a single software project.

Continuous Delivery : Continuous Delivery is the ability to get changes ofall types such as including new features, configuration changes, bug fixes
and experiments into production, or into the hands of users, safely and quickly in a sustainable way.
We use Github Actions to build our CI/CD pipeline.

First launched in 2018 as a platform-native automation tool, GitHub Actions has evolved to give developers powerful automation and
CI/CD (continuous integration/continuous deployment) capabilities right next to your code in GitHub.
At its core, GitHub Actions is designed to help simplify workflows with 16flexible automation and offer easy-to-use CI/CD capabilities built by
developers for developers.

xiii. Getting started with github actions

Github actions is a famous platform in recent times to automate developer workflows. CI/CD pipelines is one of the many workflows
offered by github to automate work processes.

xiv. Setting up github actions

All the required YAML files are placed in the .github/ directory which will be triggered on github action basis. Workflow is a collection of jobs
and these jobs will run on the trigger of an event.

xv. Github Workflows:(Build Job)

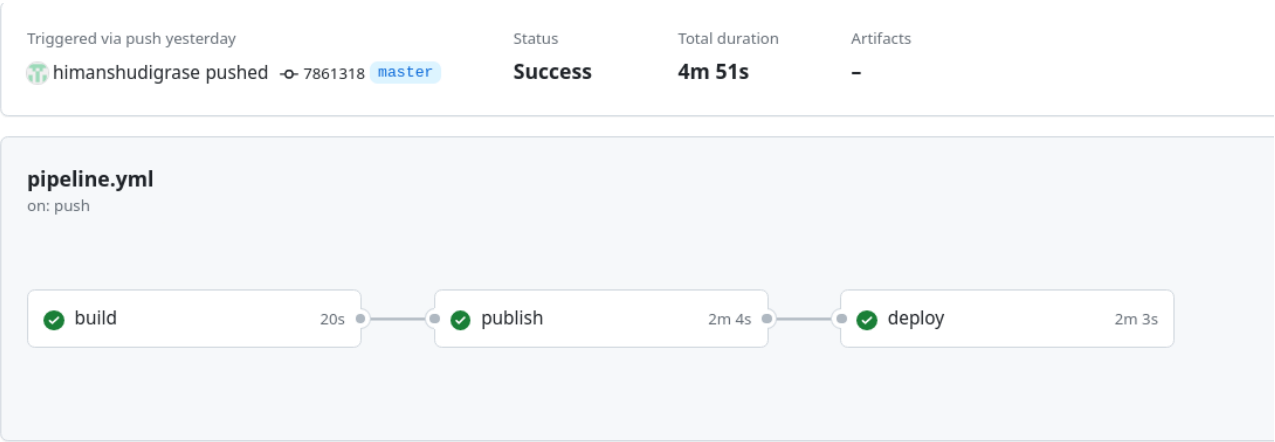

Figure: Workflow Jobs

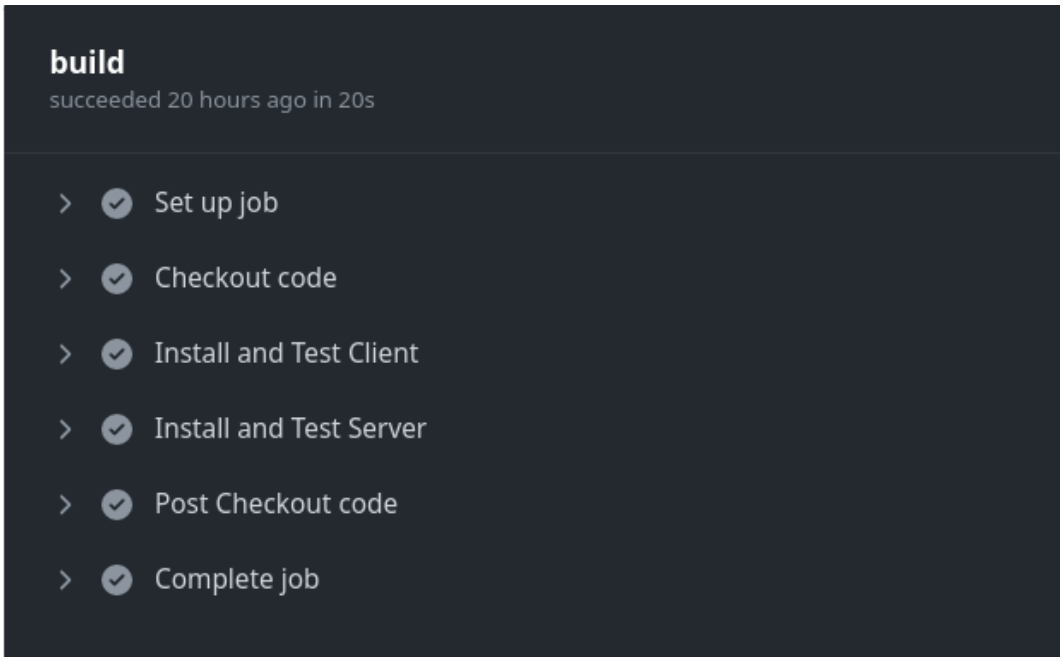

Figure: build job

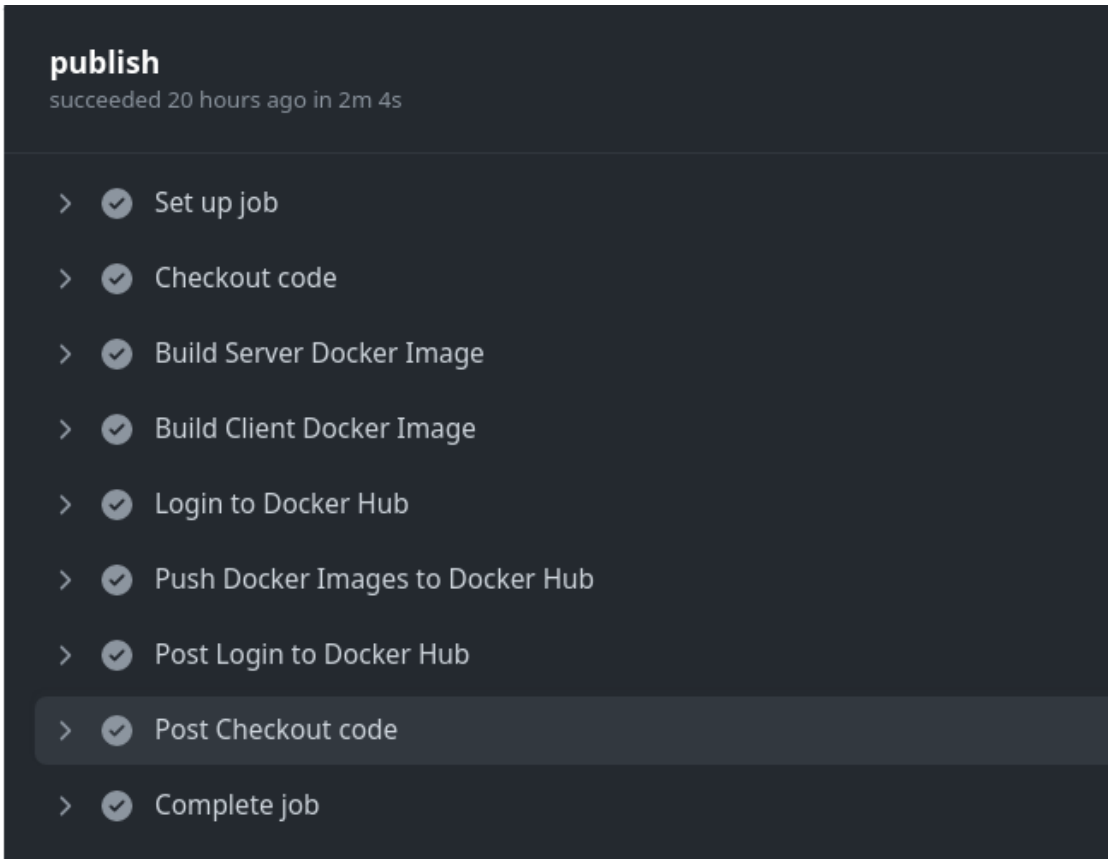

Figure: publish job

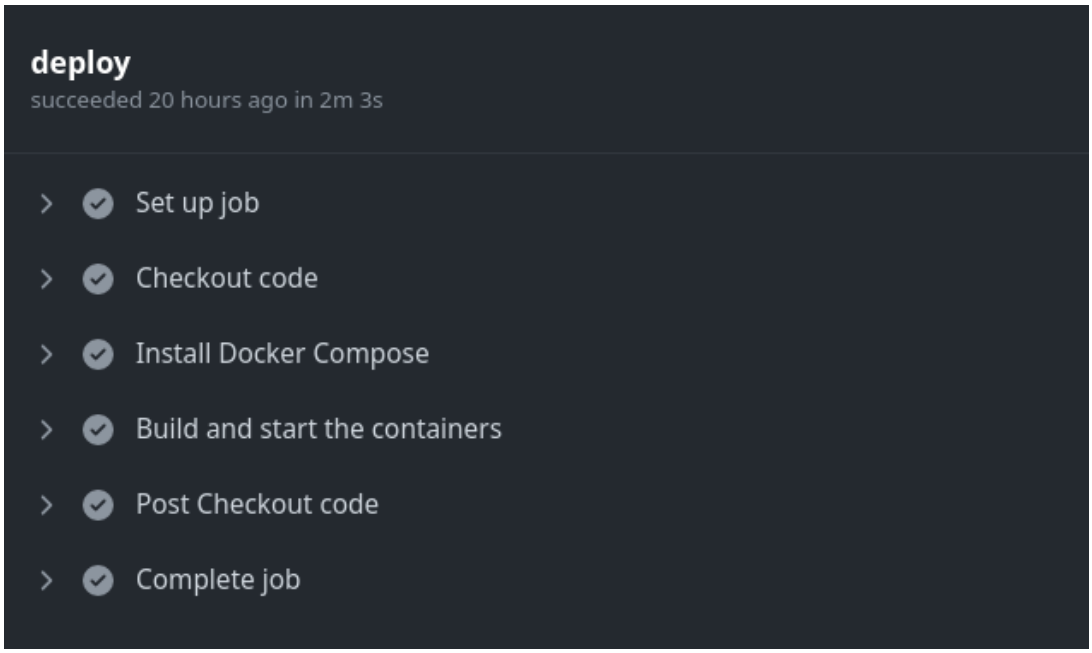

Figure: deploy job

xiv. Pipeline Script in Github Actions

Before we set up the pipeline we declare environment variables to use in the pipeline in the future. We add variables for server and client image and docker credentials.
We are storing MONGO_URL, JWT_TOKEN, PORT(Backend), PORT(Frontend) as secrets in Github. This worflow will run whenever user pushes/ commits code in Github.

```yaml
name: Build and Deploy

# Run the workflow when code is pushed to the main branch
on:
  push:
    branches:
      - master

# Set environment variables
env:
  MONGO_URL: ${{ secrets.MONGO_URL }}
  JWT_SECRET: ${{ secrets.JWT_SECRET }}
  PORT: 3001
  PORTF: 6002
```

Figure: env variables in workflow

```yaml
jobs:
  build:
    # This is telling GitHub to run the workflow on the latest version of Ubuntu.
    runs-on: ubuntu-latest
    env:
      MONGO_URL: ${{ secrets.MONGO_URL }}
      JWT_SECRET: ${{ secrets.JWT_SECRET }}
      PORT: 3001
      PORTF: 6002
    steps:
      # Checkout the code from the GitHub repository
      - name: Checkout code
        uses: actions/checkout@v3

      # Install dependencies and run tests for the client application
      - name: Install and Test Client
        working-directory: ./SocialFrontEnd/socialfrontend
        run: |
          npm install

      # Install dependencies, export environment variables to be used by application and run tests for the server application
      - name: Install and Test Server
        working-directory: ./SocialBackend
        run: |
          npm install
          export JWT_SECRET=$JWT_SECRET
          export PORT=3001
```

Figure: build job steps

Build job workflow where we are checking out the code from Github Repo to the server hosted by Github which runs ubuntu-latest. We are installling required dependencies for frontend and backend by running npm install and providing required env variables.

```yaml
  publish:
    needs: build
    runs-on: ubuntu-latest
    steps:
    # Checkout the code from the GitHub repository
      - name: Checkout code
        uses: actions/checkout@v3

      # Build a Docker image for the server application
      - name: Build Server Docker Image
        working-directory:
          ./SocialBackend

        run: |
          docker build --network host \
          --build-arg MONGO_URL=$MONGO_URL \
          --build-arg PORT=$PORT \
          --build-arg JWT_SECRET=$JWT_SECRET \
          -t hims0301/backend:latest .

      - name: Build Client Docker Image
        working-directory: ./SocialFrontEnd/socialfrontend

        run: |
          docker build --network host \
          --build-arg PORT=$PORTF \
          -t hims0301/frontend:latest .

      # Login to Docker Hub using credentials from repository secrets
      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      # Push the Docker images to Docker Hub
      - name: Push Docker Images to Docker Hub
        run: |
          docker push hims0301/frontend:latest
          docker push hims0301/backend:latest
```

Figure: publish job steps

Publish job publishes the images for frontend and backend in docker repository.It builds images of frontend and backend on server hosted by github. While building docker images we are passing the necessary env variables as arguments to docker. By logging into docker with credentials specified as secret variables we are pushing our built images to dockerHub.

```yaml
  deploy:
    needs: publish
    runs-on: self-hosted
    steps:
      # Checkout the code from the GitHub repository
      - name: Checkout code
        uses: actions/checkout@v2

      # Install Docker Compose
      - name: Install Docker Compose
        run: |
          sudo apt-get update
          sudo apt-get install -y docker-compose

      # Build and start the containers
      - name: Build and start the containers
        run: |
          docker-compose -f docker-compose.yaml up -d
```

Figure: deploy job steps

Deploy job deploys the docker images into conatiners on the localhost. We can notice that server has been specified as *self-hosted* which means our localhost is acting as server.
Docker images will be deployed on our localhost with the help of docker compose file.

Docker Compose File

docker-compose.yaml

You, yesterday | 2 authors (You and others)

```yaml
version: "3.9"
services:
  sociopedia-frontend:
    image: hims0301/frontend:latest
    container_name: sociopedia-frontend
    stdin_open: true
    # ports:
    #   - "6002:6002"
    network_mode: host
     # - sociopedia

  sociopedia-backend:
    image: hims0301/backend:latest
    container_name: sociopedia-backend
    ports:
      - "3001:3001"
    networks:
      - sociopedia
    volumes:
      - /home/himanshu/Downloads/socialLogs/logs.log:/app/logs.log
```

docker-compose.yaml

```yaml
  Elasticsearch:
    image: elasticsearch:7.16.2
    container_name: elasticsearch
    restart: always
    volumes:
    - elastic_data:/usr/share/elasticsearch/data/
    environment:
      ES_JAVA_OPTS: "-Xmx256m -Xms256m"
      discovery.type: single-node
    ports:
    - '9200:9200'
    - '9300:9300'
    networks:
      - sociopedia

  Logstash:
    image: logstash:7.16.2
    container_name: logstash
    restart: always
    volumes:
    - /home/himanshu/Downloads/socialLogs/:/logstash_dir
    command: logstash -f /logstash_dir/logstash.conf
    depends_on:
      - Elasticsearch
    ports:
    - '9600:9600'
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - sociopedia
```

docker-compose.yaml

```yaml
  Kibana:
    image: kibana:7.16.2
    container_name: kibana
    restart: always
    ports:
    - '5601:5601'
    environment:
      - ELASTICSEARCH_URL=http://elasticsearch:9200
    depends_on:
      - Elasticsearch
    networks:
      - sociopedia

networks:
  sociopedia:
    ipam:
      driver: default
      config:
      - subnet: 127.0.0.1/8
volumes:
  elastic_data:
    driver: local
```