THE SOCIAL EDGE
Dockerization of a multi-tier web application

Minor Project I

Enroll. No. (s) - 21803006, 21803013, 21803028
Name of Students - Tanya Vashistha, Vivek Shaurya, Sneha
Name of Supervisor - Dr. Amarjeet Prajapati

December - 2023

Submitted in partial fulfillment of the Degree of
5 Year Dual Degree Programme B.Tech

In

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

TABLE OF CONTENTS

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature:

Name:Tanya Vashistha

Place: Noida

Enrolment No.: 21802006

Date : 01 /12/2023

Signature:

Name: Vivek Shaurya

Enrolment No.: 21803013

Signature:

Name: Sneha

Enrolment No.: 21803028

CERTIFICATE

This is to certify the work titled "**The Social Edge**" submitted by **Tanya Vashistha**, **Vivek Shaurya** and **Sneha** in partial fulfillment of 5 year dual degree programme Btech in Computer Science Engineering from Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree or diploma.

Signature of Supervisor:

Name of Supervisor: **Dr. Amarjeet Prajapati**

Designation: Assistant Professor

Date : 01 /12/2023

ACKNOWLEDGEMENT

We would like to place on record our deep sense of gratitude to **Dr. Amarjeet Prajapati**, Associate Professor, Jaypee Institute of Information Technology, India for his generous guidance, help and useful suggestions.

We express our sincere gratitude to **Purtee Kohli, Mohit Singh, Asmita Yadav, Anuja Shukla, Amit Mishra** and **Jyoti,** Dept. of CSE and IT, Jaypee Institute of Information Technology, Noida, UP, India, for their stimulating guidance, continuous encouragement and supervision throughout the course of present work.

Signature:

Name:Tanya Vashistha

Enrolment No.: 21802006

Signature:

Name: Vivek Shaurya

Enrolment No.: 21803013

Signature:

Name: Sneha

Enrolment No.: 21803028

Date : 01 /12/2023

Summary

The culmination of our project, centered on Dockerizing a three-tier web application for a social media platform, marks a significant stride in modern software development. Embracing containerization technologies, particularly Docker, has revolutionized our approach to scalability, efficiency, and deployment. The integration of Docker with the frontend (React), backend (Node.js), and database (MySQL) components showcases a seamless, portable ecosystem. Docker Compose orchestrates this integration, streamlining deployment across diverse environments, while GitHub Actions automates our Continuous Integration/Continuous Delivery (CI/CD) pipeline. This transformative combination ensures consistent, reliable, and efficient deployment, addressing challenges prevalent in manual deployment, testing bottlenecks, inconsistent environments, dependency management, and scalability.

Our empirical study scrutinized traditional software development practices, emphasizing the inefficiencies of manual deployments, testing bottlenecks, and inconsistent environments. Docker's agility, encapsulating applications into containers, provides a definitive solution. The CI/CD pipeline, powered by GitHub Actions, further enhances the development lifecycle by automating integration, testing, and deployment. The absence of such a pipeline contributes to deployment delays, increased bug incidence, operational inefficiencies, resource wastage, and collaboration issues, which our solution effectively mitigates.

The impact of our project is evident in the tangible benefits it brings. Deployment delays are minimized, ensuring agility and responsiveness to user needs. The incidence of bugs reaching production is significantly reduced, fostering an enhanced user experience. Operational inefficiencies associated with traditional scaling methods are addressed, optimizing resource utilization. Collaboration among development and operations teams is streamlined through a standardized CI/CD pipeline.

In conclusion, the marriage of Dockerization and a robust CI/CD pipeline fortifies our social media platform's development life cycle. The project not only addresses contemporary challenges in software development but also sets the foundation for an agile, scalable, and collaborative future. The efficient deployment of our containerized application underscores the project's success, exemplifying a paradigm shift in how we approach, build, and deploy complex, multi-tiered web applications.

CHAPTER-1
INTRODUCTION

1.1 GENERAL INTRODUCTION:

In the rapidly evolving realm of web development, the relentless pursuit of scalable, efficient, and easily deployable applications has spurred a paradigm shift – the widespread adoption of containerization technologies. At the forefront of this transformative wave stands our project: a groundbreaking social media website that serves as a testament to the remarkable power and versatility of Docker.

Docker, a preeminent containerization platform, has redefined how applications are developed, tested, and deployed. It facilitates the encapsulation of applications and their dependencies into lightweight, portable containers, ensuring a level of consistency across various environments that was once an elusive goal. Our social media project, a multi-faceted endeavor incorporating frontend (React), backend (Node.js), and database (MySQL) components, exemplifies the seamless integration and orchestration achieved through Docker, Docker Compose, and GitHub Actions.

The journey commences with an exploration of containerization's transformative influence. Docker emerges as the linchpin, providing the means to encapsulate our multi-tiered social media project into containers that transcend the constraints of individual environments. This ensures a harmonious and reliable execution of our application, regardless of the development stage or deployment environment.

Delving into the frontend development intricacies, React, a dynamic JavaScript library for building interactive user interfaces, takes center stage. Its component-based architecture not only enhances modularity and reusability but also aligns seamlessly with Dockerization, eradicating the notorious "it works on my machine" dilemma. Meanwhile, the backend, fortified by Node.js and MySQL, undergoes a transformation through Dockerization, unlocking the potential for easy

scaling, versioning, and deployment. Node.js's non-blocking, event-driven architecture optimizes performance, while MySQL's reliability and robustness effortlessly manage our social media platform's data dynamics.

The orchestration of these components reaches its zenith with Docker Compose, a powerful tool that simplifies deployment complexities by defining services, networks, and volumes in a single YAML file. This ensures the consistent deployment of our entire application stack across diverse environments, from development to testing to production.

Complementing these advancements is the pivotal role of GitHub Actions in establishing a Continuous Integration/Continuous Delivery (CI/CD) pipeline. This transformative process automates building, testing, and deploying our application with every code change, ushering in a new era of reliability, efficiency, and collaboration in our development workflow.

In the crucible of these technologies and methodologies, our social media project emerges not just as a technological marvel but as a beacon illuminating the path toward agile development, rapid deployment, and the creation of scalable, consistent environments. This amalgamation of Docker, React, Node.js, Docker Compose, and GitHub Actions not only addresses the challenges of modern software development but propels our venture into a future where collaboration, efficiency, and reliability define the new standard.

1.2 PROBLEM STATEMENT:

The challenge at hand revolves around the containerization of a social media website, employing Docker for frontend (React), backend (Node.js), and database (MySQL) components. Integration of Docker Compose streamlines deployment, while GitHub Actions establishes a Continuous Integration/Continuous Delivery (CI/CD) pipeline. In the absence of these advancements, the project grapples with manual deployment complexities, leading to time-consuming processes and the notorious "it works on my machine" scenario. Testing bottlenecks arise without the aid of automated CI/CD pipelines, hindering bug detection and resolution. Inconsistent environments pose a hurdle, causing unexpected behaviors and impeding collaboration among team members. Traditional dependency management results in dependency hell, and scaling applications sans

containerization proves resource-intensive. The cumulative impact manifests as deployment delays, increased bug incidence, operational inefficiencies, resource wastage, and collaboration issues, underscoring the imperative need for Dockerization and CI/CD pipelines in modern software development.

1.3 SIGNIFICANCE OF THE PROBLEM:

This project stands as a strategic and adept response to the key challenges prevalent in modern software development. It confronts and effectively addresses critical issues, including the intricacies of manual deployment, bottlenecks in testing processes, challenges posed by inconsistent environments, the complexities of managing dependencies, and the hurdles associated with scaling applications. The adoption of Docker and the implementation of Continuous Integration/Continuous Delivery (CI/CD) pipelines form the cornerstone of our transformative approach.

Dockerization offers a solution to the age-old challenge of manual deployment by encapsulating applications and their dependencies into portable containers. This ensures consistency across diverse environments, tackling the notorious "it works on my machine" scenario and streamlining the deployment process.

The implementation of CI/CD pipelines, powered by robust tools like GitHub Actions, tackles testing bottlenecks by automating integration, testing, and deployment processes. This not only accelerates the development lifecycle but also enhances reliability, ensuring early detection of bugs and reducing the likelihood of manual errors.

Addressing inconsistent environments, our project leverages Docker's capability to create containers, providing a consistent and reproducible environment throughout the development lifecycle. This mitigates unexpected behavior across different stages of development and fosters collaboration among team members.

The challenge of dependency management is effectively handled through Docker, which encapsulates dependencies within containers, eliminating the chaos of version conflicts and the infamous "dependency hell."

Scaling applications is simplified through Docker's agility, enabling easy scaling, versioning, and deployment. This addresses scalability challenges in a resource-efficient manner, catering to the dynamic demands of a social media platform.

The overall impact of this project is a significant enhancement in efficiency, reliability, and collaborative synergy throughout the software development lifecycle. By strategically addressing these challenges, our project not only overcomes obstacles inherent in modern software development but also contributes to the advancement of software engineering practices. This commitment to innovation positions our project at the forefront of the evolving landscape of web development, reflecting the transformative potential of Docker and CI/CD pipelines in shaping the future of software development.

1.4 EMPIRICAL STUDY:

The empirical study dissected traditional challenges in three-tier web application development, spotlighting manual deployment intricacies, testing bottlenecks, inconsistencies in environments, dependency management pitfalls, and scalability challenges. It revealed the drawbacks of conventional tools and methodologies. In contrast, Dockerization proved transformative, ensuring consistency across diverse environments, while Continuous Integration/Continuous Delivery (CI/CD) pipelines automated processes, enhancing efficiency and reliability. This empirical exploration underscores the necessity of embracing innovative solutions like Docker and CI/CD pipelines for a streamlined, scalable, and efficient three-tier web application development lifecycle.

1.5 SOLUTION APPROACH :

At the core of our innovative solution lies the strategic adoption of Docker, serving as the keystone for comprehensive containerization within our social media platform. This deliberate choice manifests in the meticulous encapsulation of every element, from the frontend developed in React to the backend powered by Node.js and the robust MySQL database. Docker's prowess in creating lightweight, portable containers ensures the consistent execution of our application

across diverse environments, be it during development, testing, or in the demanding production landscape.

Central to our containerization strategy is Docker Compose, a powerful orchestration tool that takes center stage in harmonizing the integration of the frontend, backend, and database components. This orchestration process is pivotal, facilitating a seamless deployment experience. The use of a single YAML file to define services, networks, and volumes streamlines the setup, ensuring that our entire application stack is deployed consistently. Docker Compose plays a crucial role in unifying the deployment process, underscoring its significance in achieving a uniform and reliable deployment across various environments.

Complementing Docker and Docker Compose, GitHub Actions emerges as a linchpin in automating our Continuous Integration/Continuous Delivery (CI/CD) pipeline. This automated workflow is engineered to handle the entire spectrum of development processes, from the introduction of code changes to the deployment of the application. GitHub Actions acts as the driving force behind our agile development practices, reducing manual intervention and fostering a dependable, error-resistant development lifecycle. The integration of GitHub Actions is not merely a technological facet; it signifies a commitment to efficiency, collaboration, and a streamlined development workflow.

The synergistic interplay of Docker, Docker Compose, and GitHub Actions shapes a robust framework that propels our social media project into an era characterized by agile development, consistency, and dependable deployment practices. This strategic amalgamation addresses the complex challenges associated with multi-tiered web application development. The Dockerized ecosystem ensures that our application remains portable, scalable, and resilient across the various stages of its lifecycle.

Moreover, the integration of GitHub Actions for CI/CD enhances collaboration among team members, fostering a culture of efficiency and reliability. The automated pipeline significantly reduces deployment delays, ensuring that new features and updates can be swiftly released to meet the dynamic needs of our users and respond to market trends promptly.

In essence, our solution doesn't merely embrace containerization; it transforms the entire development and deployment landscape. Docker, Docker Compose, and GitHub Actions converge to create an environment where consistency, scalability, and reliability are not just aspirations but tangible outcomes. As we navigate the ever-evolving landscape of web development, this integrated approach positions our social media project at the forefront of modern, efficient, and agile software development practices.

1.6 EXISTING APPROACH TO THE PROBLEM FRAME:

Conventional software development grapples with manual deployment intricacies, testing bottlenecks, environmental disparities, dependency complexities, and scalability challenges. In stark contrast, our innovative approach seamlessly integrates Docker for containerization and leverages GitHub Actions for Continuous Integration/Continuous Delivery (CI/CD). This fusion establishes a streamlined, automated, and uniform development lifecycle. The juxtaposition highlights the transformative advantages of embracing containerization and CI/CD pipelines, positioning our methodology as a forward-looking solution to the persistent challenges encountered in modern software development practices.

CHAPTER-2
LITERATURE SURVEY

2.1 SUMMARY OF THE PAPER STUDIED:

1. Developing Docker and Docker-Compose Specifications: **A** Developers' Survey
   https://ieeexplore.ieee.org/abstract/document/9658534

SUMMARY

This article explores insights into the development of container and orchestration specifications, offering valuable complements to Guerriero et al.'s general Infrastructure as Code (IaC) interviews. Distinctively, our study delves into specific activities related to developing specifications for two IaC technologies—Docker and Docker-Compose. Findings reveal that the Dockerfile development process presents challenging and time-consuming activities, with identifying system dependencies and debugging emerging as particularly critical areas for improvement. While Docker-Compose results were less pronounced, debugging activities remain a common concern, especially for less-experienced developers. Notably, most developers do not incorporate ancillary tools into their development processes. This work suggests avenues for future research, proposing a usability evaluation to identify major bottlenecks and exploring new approaches or environments to address time-consuming activities. Concepts like liveness and live software development are envisioned to play a pivotal role, proactively providing feedback to users. Additionally, visual programming and model-driven engineering techniques are considered crucial, especially for less-experienced developers, representing a promising direction for advancing the development of Docker and Docker-Compose specifications.

2. Performance Evaluation of Docker Container and Virtual Machine

https://www.sciencedirect.com/science/article/pii/S1877050920311315?ref=pdf_download&fr=RR-2&rr=82ea11914bbc2af8

SUMMARY

This paper extensively examines the widespread usage of virtual machines (VMs) in enterprise scenarios, highlighting their versatility in handling diverse operations from Hadoop tasks to smaller applications. Recognizing the inherent inefficiencies in VMs, the paper advocates for a lightweight solution, leading to the emergence of Docker containers as an efficient and swift virtualization technology. Related works on VMs and Docker performance evaluations are referenced, concentrating on crucial metrics like CPU performance, memory throughput, and operational speed. Employing benchmarking tools, the evaluation comprehensively compares KVM and Docker on two HP servers. Results consistently underscore Docker containers' superior performance across all parameters compared to virtual machines, attributed to the inefficiencies associated with the QEMU layer in VMs. The paper concludes by outlining future directions, expressing a focus on exploring container scheduling in Docker and developing more secure container variants to address security concerns, providing valuable insights into Docker's superior performance in enterprise environments.

2.2 INTEGRATED SUMMARY OF THE LITERATURE STUDIED:

The literature examined encompasses two distinct but complementary aspects of contemporary software development practices. The first study, "Developing Docker and Docker-Compose Specifications: A Developers' Survey," sheds light on the nuanced challenges and practices involved in the development of container and orchestration specifications. Focusing on Docker and Docker-Compose, the study identifies challenging and time-consuming aspects within the Dockerfile development process, particularly emphasizing the critical areas of identifying system dependencies and debugging. Notably, the study proposes avenues for future research, advocating for a usability evaluation to address major bottlenecks and exploring innovative approaches, such as liveness and live software development, to enhance user feedback.

The second study, "Performance Evaluation of Docker Container and Virtual Machine," critically evaluates the prevalent use of virtual machines (VMs) in enterprise settings and advocates for the adoption of Docker containers as a lightweight and efficient virtualization technology. Concentrating on crucial metrics such as CPU performance, memory throughput, and operational speed, the study benchmarks KVM and Docker on two HP servers. Results consistently demonstrate the superior performance of Docker containers across all parameters compared to virtual machines, highlighting the inefficiencies associated with the QEMU layer in VMs. The paper concludes by outlining future directions, including the exploration of container scheduling in Docker and the development of more secure container variants to address security concerns.

Together, these studies provide a comprehensive perspective on modern software development practices, from the intricate challenges faced during the development of container specifications to the performance advantages offered by Docker containers over traditional virtual machines in enterprise environments. The integrated summary underscores the evolving landscape of Infrastructure as Code (IaC) technologies and emphasizes the ongoing efforts to enhance efficiency, security, and usability in the development and deployment of software applications.

CHAPTER-3
REQUIREMENT ANALYSIS AND SOLUTION APPROACH

3.1 REQUIREMENT ANALYSIS:

1.FUNCTIONAL REQUIREMENTS:

Containerization: The system should facilitate the containerization of the frontend (React), backend (Node.js), and database (MySQL) components through Docker. This includes ensuring seamless deployment and interaction of these components within isolated containers.

Continuous Integration (CI): Implement CI pipelines to automate the build, test, and validation processes for both React and Node.js components. The CI system should provide real-time feedback on code changes to maintain code quality and consistency.

Continuous Delivery (CD): Establish CD pipelines for the automated deployment of containerized applications to different environments. This includes the orchestration of the deployment process, ensuring that the applications are consistently delivered to various stages, such as development, testing, and production.

Database Management: Ensure the proper containerization and management of the MySQL database. This involves creating Docker containers specifically tailored for the MySQL database, guaranteeing efficient data storage, retrieval, and consistent behavior across different deployment environments.

2. NON- FUNCTIONAL REQUIREMENTS:

Scalability: The solution must exhibit scalability, effectively accommodating fluctuating levels of user activity. This applies particularly to the backend Node.js server and the MySQL database, ensuring the system's responsiveness under varying workloads.

Portability: Utilizing Docker containers, the application should demonstrate high portability, enabling consistent and reliable execution across diverse environments. This ensures seamless deployment and operation regardless of the underlying infrastructure.

Reliability: The CI/CD pipelines must contribute to the overall reliability of the system by automating testing and deployment procedures. This reduces the probability of errors in production, enhancing the stability and dependability of the deployed applications.

Efficiency: Dockerization and the incorporation of CI/CD practices should optimize resource usage. This efficiency extends to both development and deployment processes, promoting resource-conscious practices that contribute to faster, more streamlined workflows.

3. SECURITY REQUIREMENTS:

Container Security: The system must incorporate robust security measures for Docker containers, ensuring their isolation and mitigating potential security risks. This includes implementing measures to prevent unauthorized access and safeguard the integrity of the containerized components.

Authentication and Authorization: Secure authentication and authorization mechanisms must be implemented to regulate user access to the social media platform. This involves ensuring that only authorized users can interact with the system, thereby enhancing overall security and protecting user data..

3.2 SOLUTION APPROACH:

Containerization with Docker:

Leverage Docker to encapsulate the frontend, backend, and database components, ensuring a consistent environment across diverse settings. This approach facilitates efficient dependency management and promotes uniformity in deployment.

Continuous Integration (CI):

Employ CI pipelines through tools like GitHub Actions, automating build and test processes for React and Node.js components. This proactive approach enables early bug detection and ensures code quality with each modification.

Continuous Delivery (CD):

Institute CD pipelines to automate the deployment of containerized applications across various environments (e.g., development, testing, production). This encompasses the seamless deployment of frontend, backend, and database components.

Scalability and Resource Efficiency:

Harness Docker's scalability features to dynamically scale the backend Node.js server as per demand. Implement robust resource monitoring and management mechanisms to optimize resource utilization efficiently.

Version Control and Rollback:

Integrate version control systems like Git with CI/CD pipelines to meticulously track code changes. This integration facilitates straightforward rollbacks in case of deployment issues or critical bugs, ensuring version control and system stability

Security Measures:

Enforce best practices for container security, safeguarding Docker containers against potential threats and ensuring isolation. Additionally, integrate secure authentication and authorization mechanisms to regulate user access to the social media platform, enhancing overall system security.

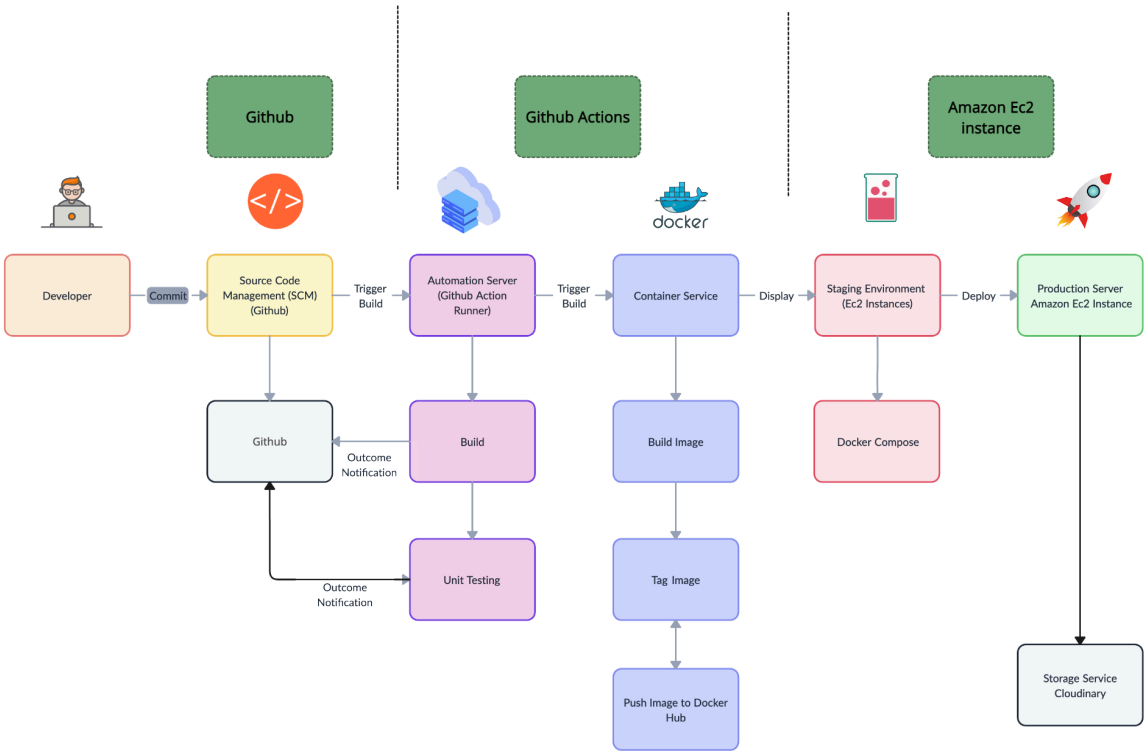
Documentation and Training:

Provide comprehensive documentation for developers, offering guidance on Docker usage, CI/CD pipeline configurations, and system architecture. Conduct training sessions to familiarize development and operations teams with the newly implemented CI/CD and Dockerized workflows, ensuring a smooth transition and optimal utilization of the new processes.

CHAPTER-4
MODELING AND IMPLEMENTATION DETAILS

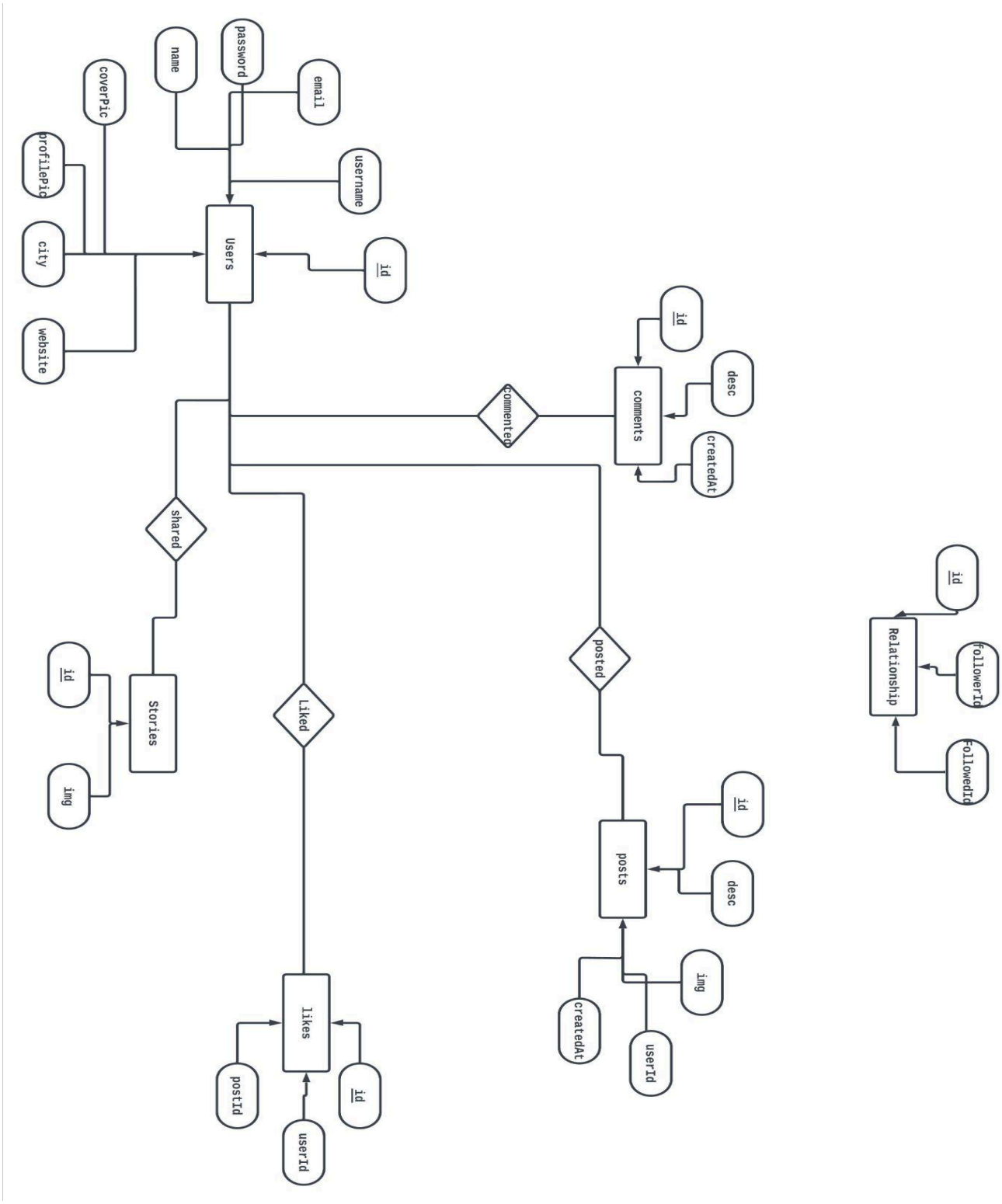
4.1 DESIGN DIAGRAMS

Workflow Diagram:

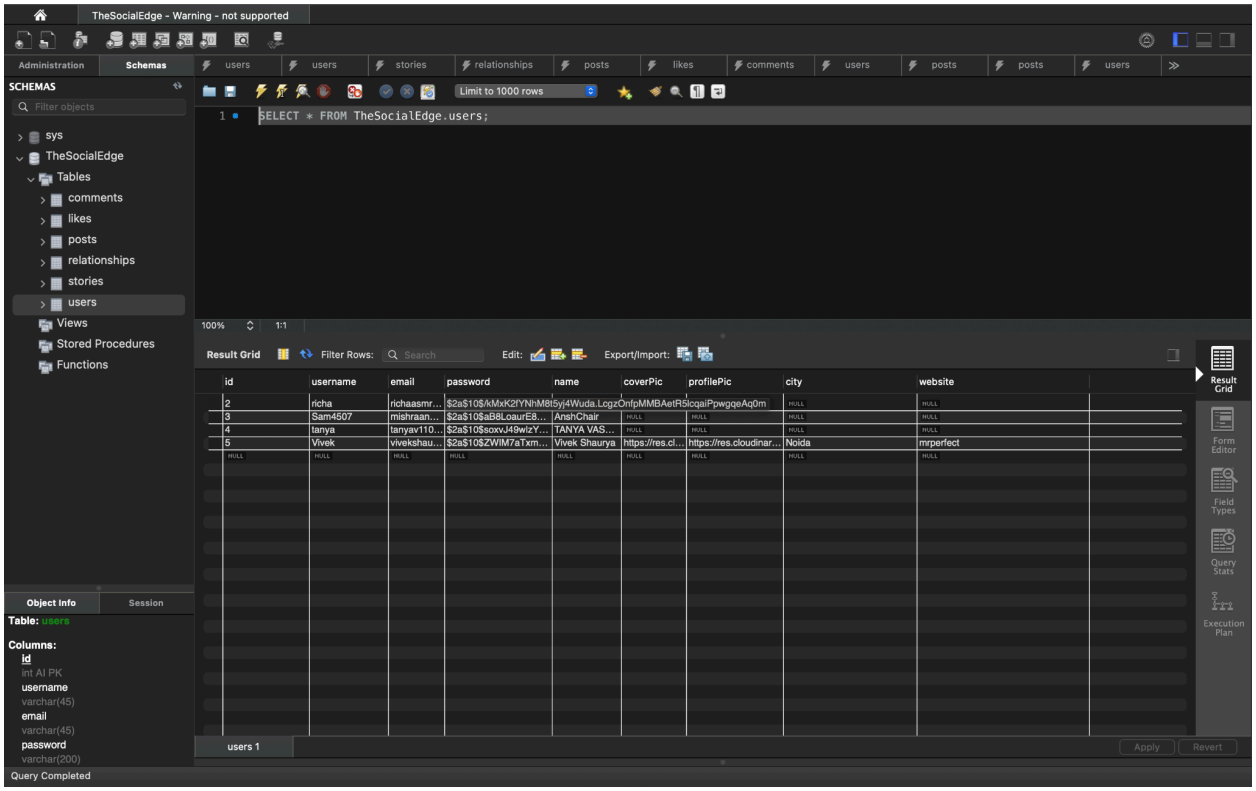


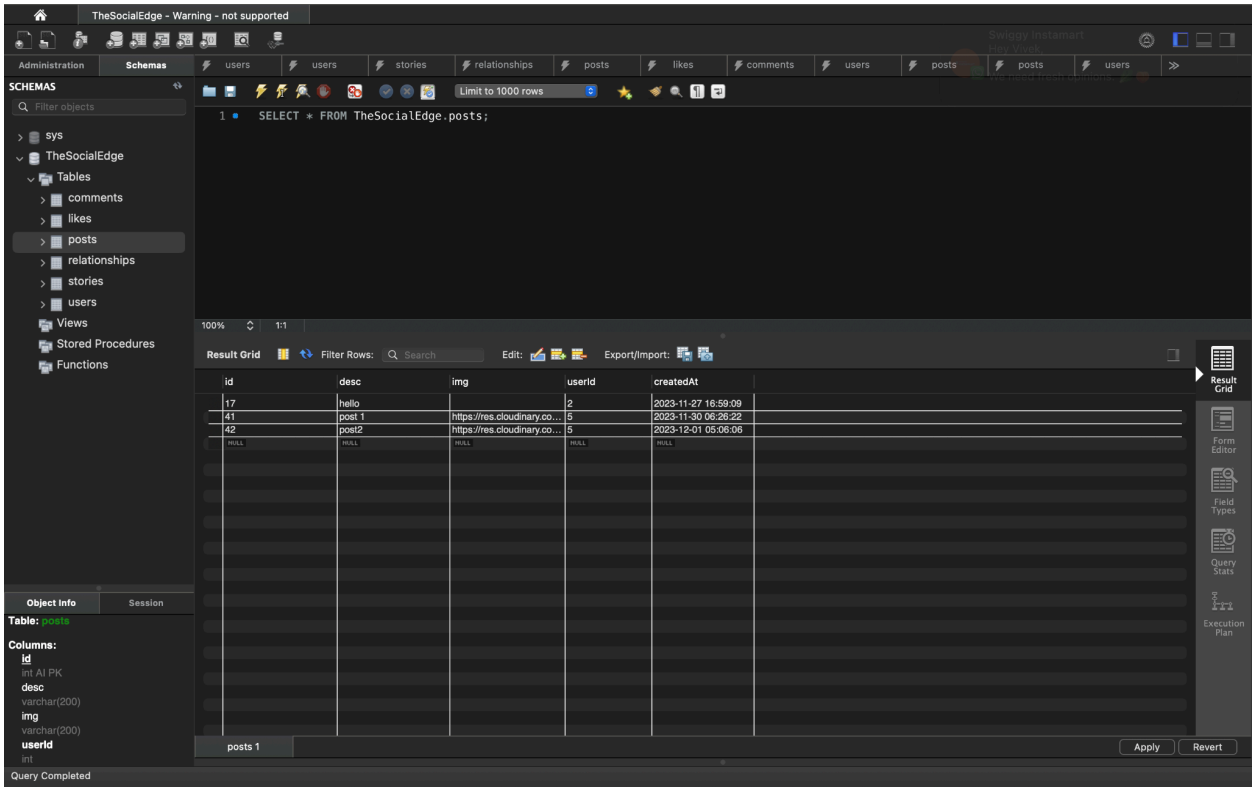
The Continuous Integration/Continuous Delivery (CI/CD) pipeline for TheSocialEdge backend is a seamless and automated process that begins when code is pushed to the main branch. Upon the push event, a GitHub Actions runner is triggered, initiating the CI/CD workflow. The workflow orchestrates the build process on an EC2 instance, leveraging the power of Docker to encapsulate the entire project. The codebase is compiled, tested, and packaged into a Docker image, which is then securely pushed to a private Docker repository. Subsequently, another EC2 server, responsible for hosting the live site, pulls the latest Docker image and deploys it by running containers. This streamlined pipeline ensures consistent, reliable, and efficient deployment, reducing manual intervention and enhancing the development workflow for TheSocialEdge backend.

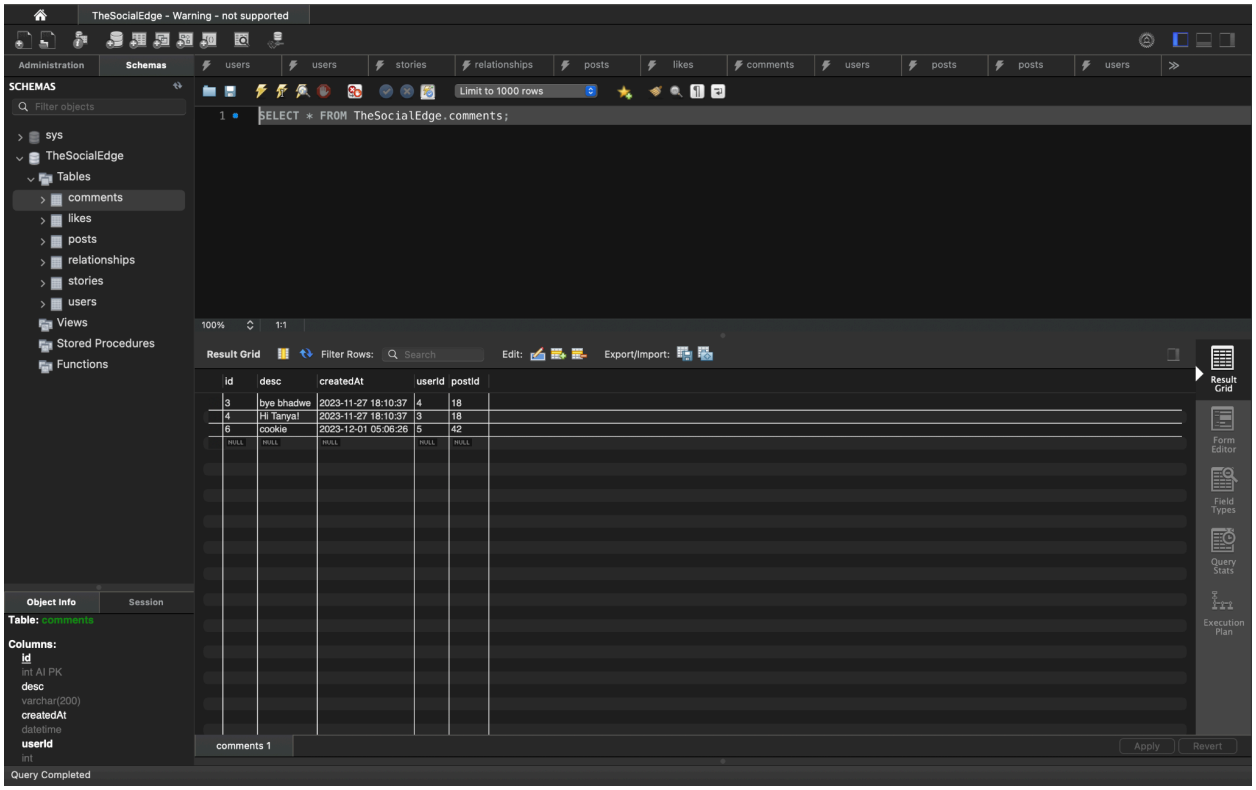
ER diagram:

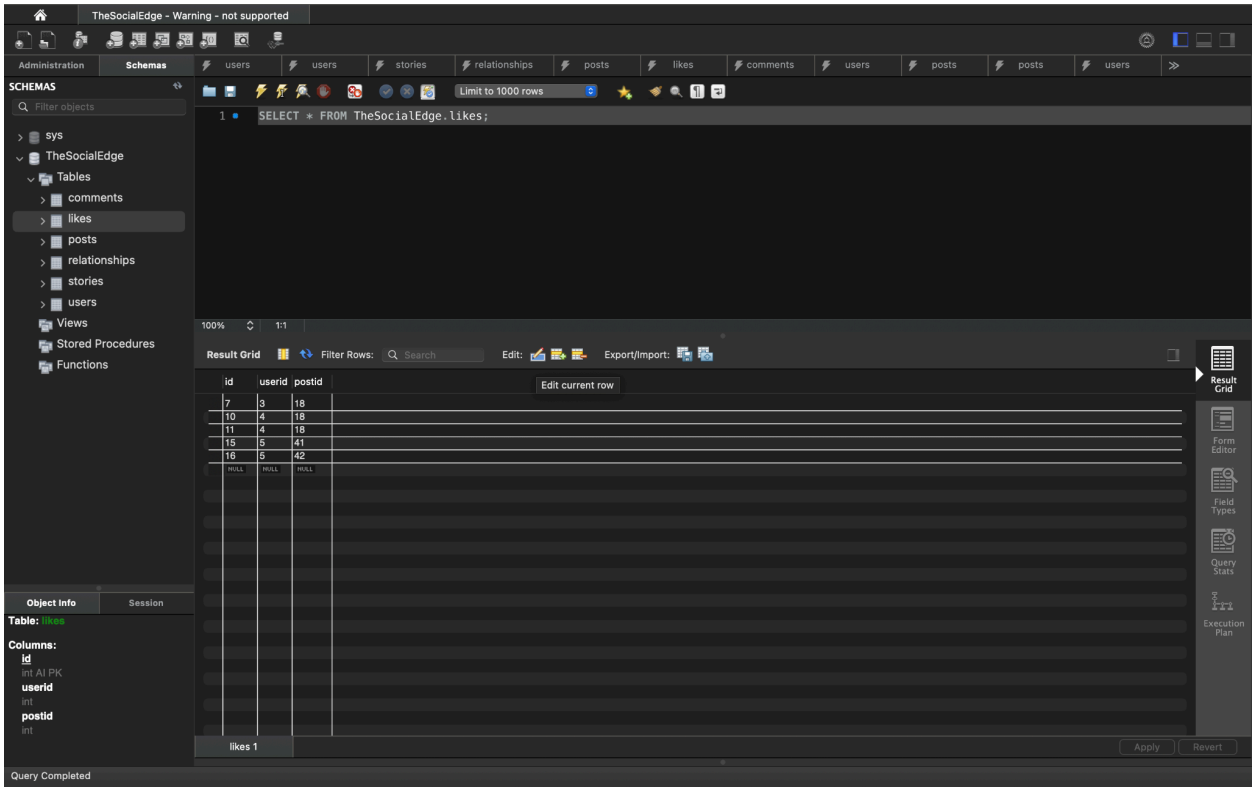

name
password
email
coverPic
profilePic
username
city
Users
id
website
id
desc
comments
createdAt
commented
shared
posted
id
Stories
img
Liked
id
desc
posts
img
createdAt
userId
likes
postId
id
userId
id
Relationship
followerId
followedId

MySQL DATABASE:






TheSocialEdge - Warning - not supported
SELECT * FROM TheSocialEdge.comments;
id desc createdAt userId postId
3 bye bhadwe 2023-11-27 18:10:37 4 18
4 Hi Tanya! 2023-11-27 18:10:37 3 18
6 cookie 2023-12-01 05:06:26 5 42
Table: comments
Columns:
id
int AI PK
desc
varchar(200)
createdAt
datetime
userId
int
Query Completed


TheSocialEdge - Warning - not supported
SELECT * FROM TheSocialEdge.likes;
id userid postid
7 3 18
10 4 18
11 4 18
15 5 41
16 5 42
Table: likes
Columns:
id
int AI PK
userid
int
postid
int
Query Completed

4.2 IMPLEMENTATION DETAILS

DEVELOPING CI/CD PIPELINE:

Developing the CI/CD pipeline for our social media website involved leveraging GitHub Actions, Action Runners, and an AWS EC2 instance. The following succinctly outlines the key steps in the development process:

1. GitHub Actions Setup:

Initiated by configuring GitHub Actions in the repository. Defined workflows in YAML files to outline the sequence of steps for CI/CD processes.

2. Workflow Triggers:

Established triggers for workflows, linking them to specific events, such as pushes to the repository, pull requests, or releases. This ensured automatic execution of workflows upon relevant events.

3. Build and Test Stage:

Incorporated build and test stages in the CI workflow. Utilized Node.js and React scripts to build the frontend and backend components, running unit tests to ensure code integrity.

4. Dockerization:

Integrated Docker within the workflow to containerize the application components. Defined Dockerfiles to encapsulate dependencies and ensure consistency across environments.

5. AWS EC2 Instance as Action Runner:

Deployed an EC2 instance on AWS to act as an Action Runner, providing scalable compute resources for GitHub Actions. Configured the runner to execute workflows on the EC2 environment.

6. Deployment Stage:

Configured deployment stages in the CD workflow. Employed AWS CLI commands within GitHub Actions to deploy Docker containers to the AWS EC2 instance.

7. Environment Configuration:

Ensured that necessary environment variables and secrets were securely configured within GitHub Actions, preventing sensitive information exposure.

8. Workflow Visualization and Monitoring:

Leveraged GitHub Actions' visualization tools to monitor workflow progress. Utilized logs and status checks to identify and troubleshoot any issues during the CI/CD pipeline execution.

9. Parallelization and Optimization:

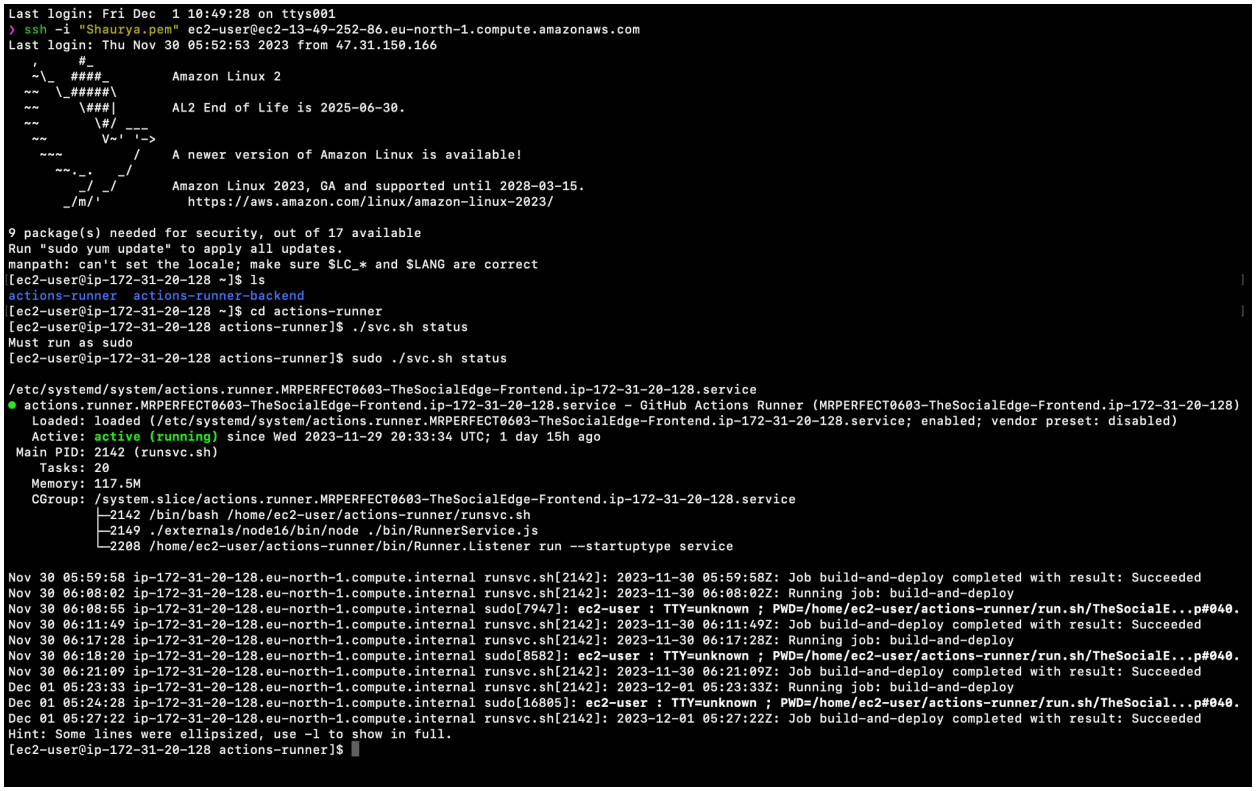
Optimized workflows by parallelizing tasks where possible, minimizing build times, and enhancing overall pipeline efficiency.

10. Notification and Reporting:

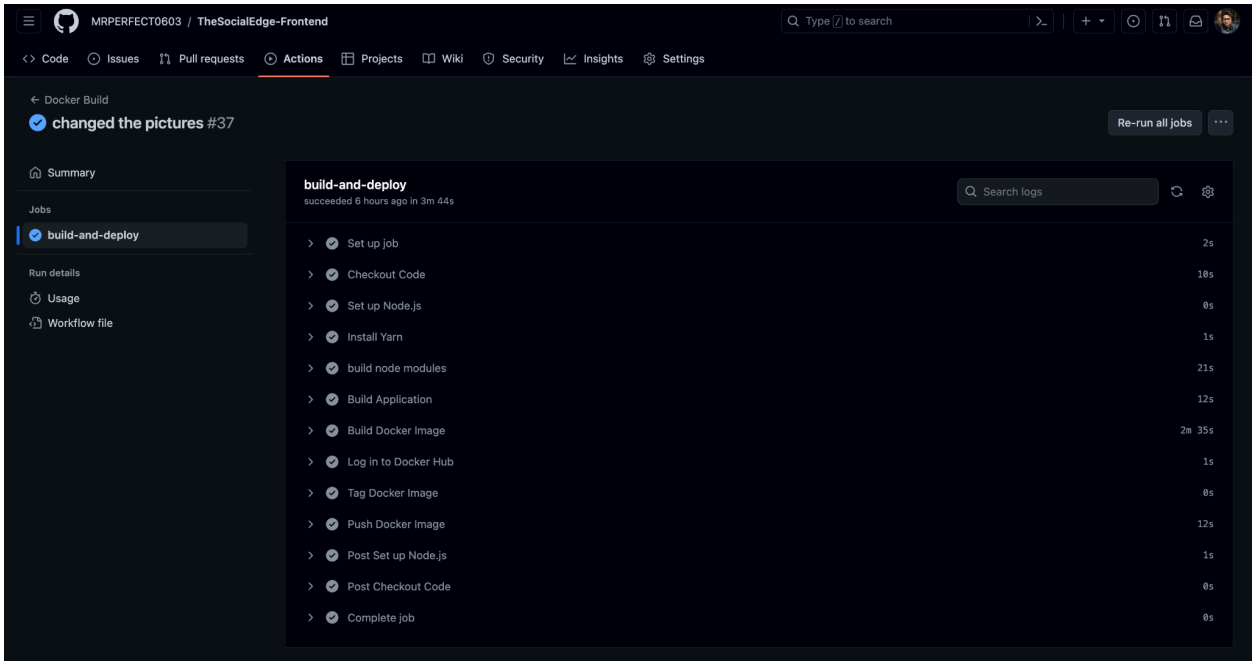
Configured notifications within GitHub Actions to alert the team about workflow completion, success, or failure. This facilitated real-time awareness and collaboration.

By implementing GitHub Actions, Action Runners on AWS EC2, and Dockerization, we achieved a robust CI/CD pipeline. This automated workflow ensures the seamless integration, testing, and deployment of our social media website, promoting rapid development cycles and reliable releases while utilizing scalable computing resources on AWS.

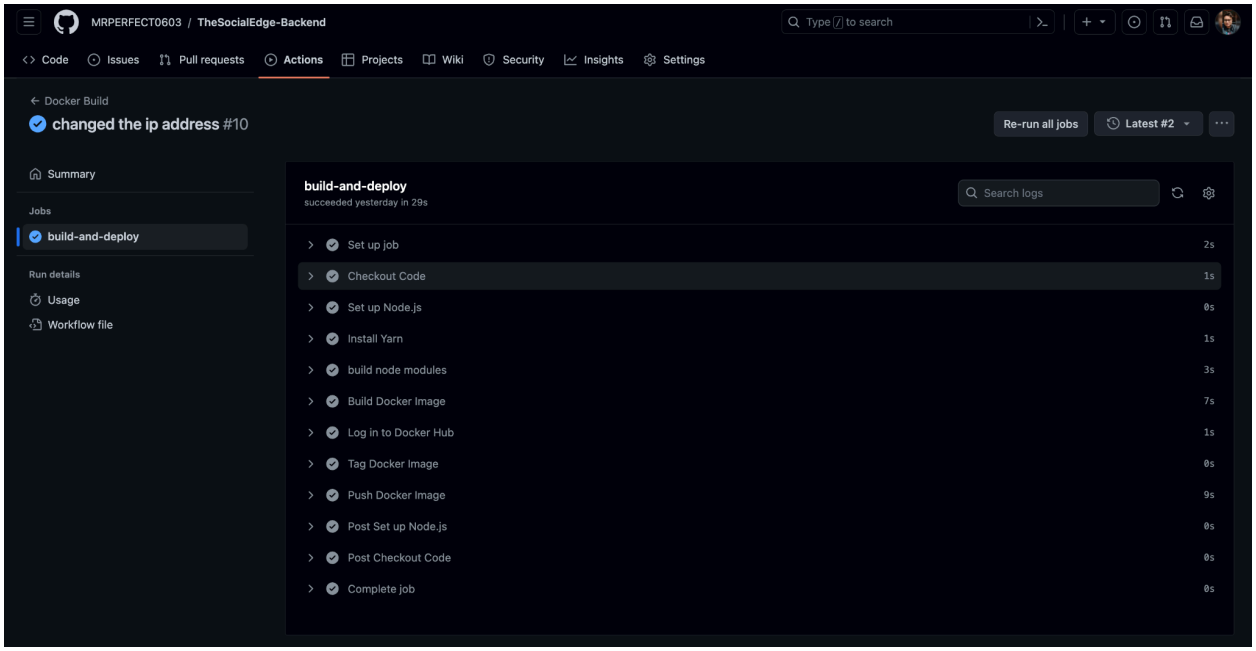
ACTION RUNNER:



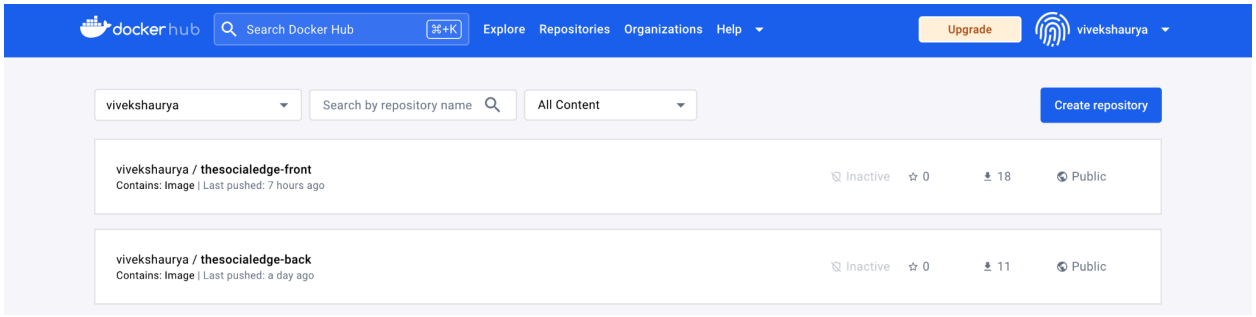
FRONT -END REPOSITORY:



BACK-END REPOSITORY:



DOCKER HUB REPOSITORY:



DOCKERIZING OUR WEB-APP:

Dockerizing our social media website involved leveraging Docker and Docker Hub to encapsulate the application components and streamline the deployment process. Here's a step-by-step explanation of how we achieved this:

1. Docker Installation:
Installed Docker on the local development machine and any other environments where the application would be deployed. This provided the necessary runtime environment to create and run Docker containers.

2. Dockerfile Creation:
Created Dockerfiles for each application component—frontend (React), backend (Node.js), and the MySQL database. Dockerfiles specified the base images, dependencies, and commands needed to set up and run each component within a Docker container.

3. Docker Compose for Multi-Container Orchestration:
Utilized Docker Compose to define the multi-container application stack. The `docker-compose.yml` file specified the services, networks, and volumes required for orchestrating the frontend, backend, and database containers.

4. Testing Locally:
Ran the Docker Compose configuration locally to ensure that the containers interacted correctly, simulating the actual production environment. This step helped identify and address any issues related to container communication or configuration.

5. Deployment with Docker Compose:

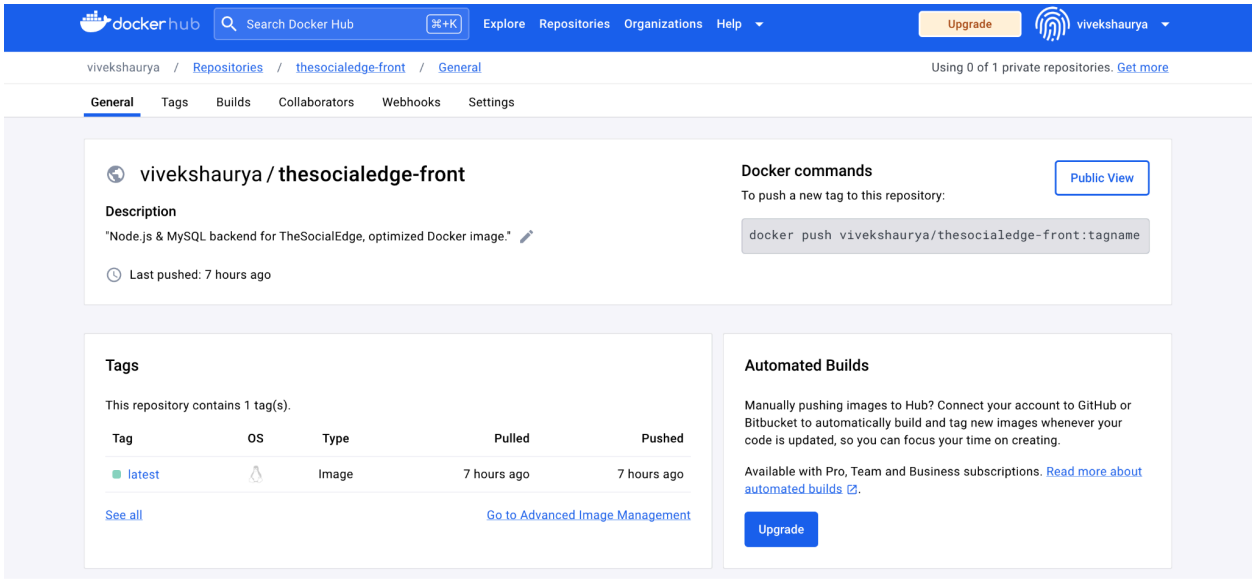
Integrated Docker Hub into the deployment process by configuring the `docker-compose.yml` file to pull the latest Docker images from Docker Hub during deployment. This ensured that the deployed containers used the most up-to-date images.

6. Automation with CI/CD:

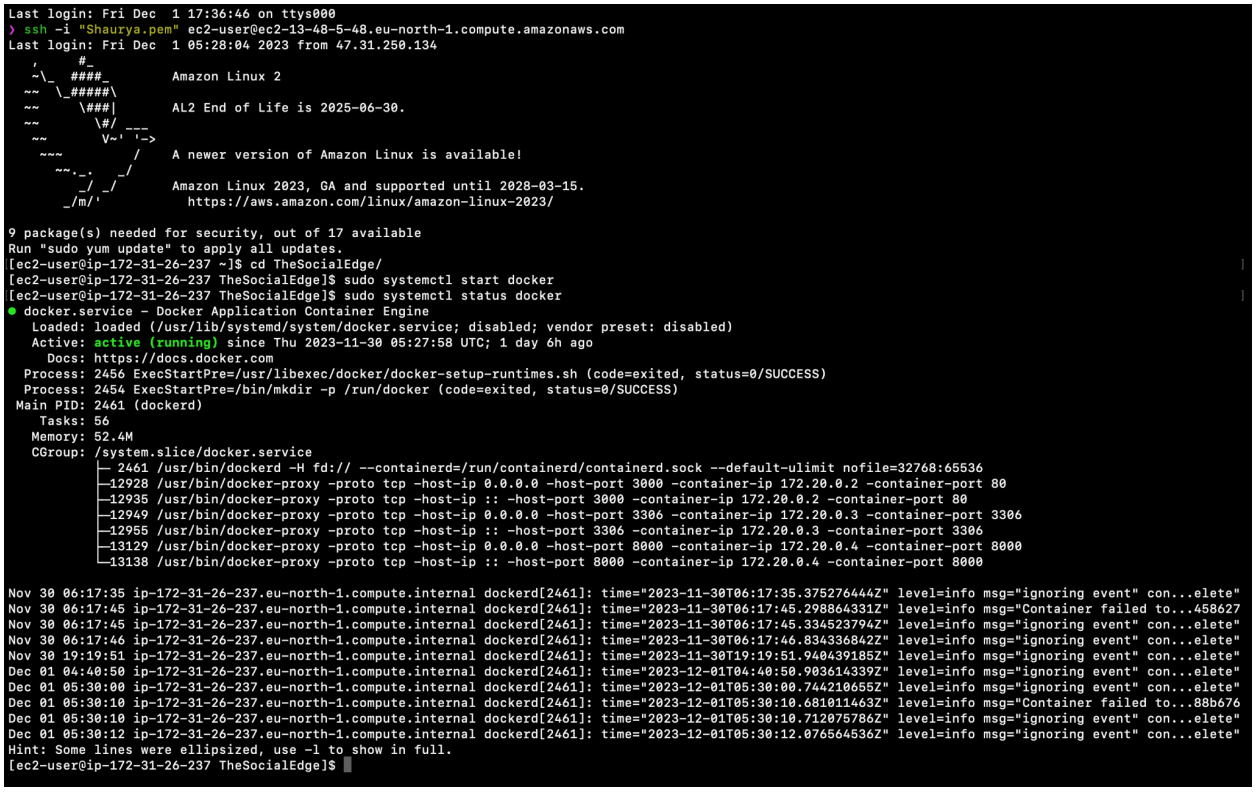
Integrated Docker image building and pushing Docker Hub into the CI/CD pipeline using tools like GitHub Actions. This automated the process, triggering image updates upon code changes and ensuring consistent, reliable deployments.

By Dockerizing our application components and utilizing Docker Hub, we achieved a containerized, portable, and scalable solution. This approach enhances consistency, facilitates collaboration, and simplifies deployment across diverse environments, ultimately contributing to a more efficient and reliable development and deployment process for our social media website.

DOCKER IMAGE OF FRONT-END:



DOCKER ON AWS SERVER:

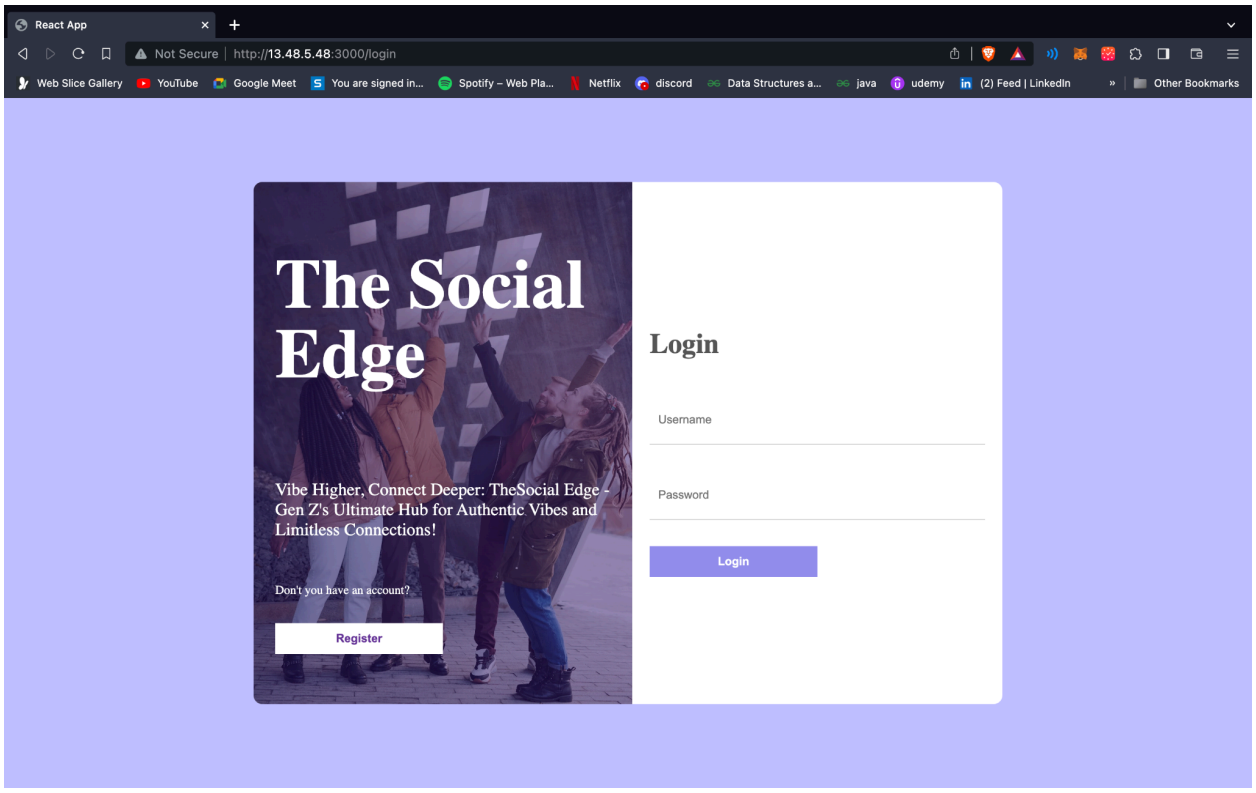


DOCKER IMAGES:

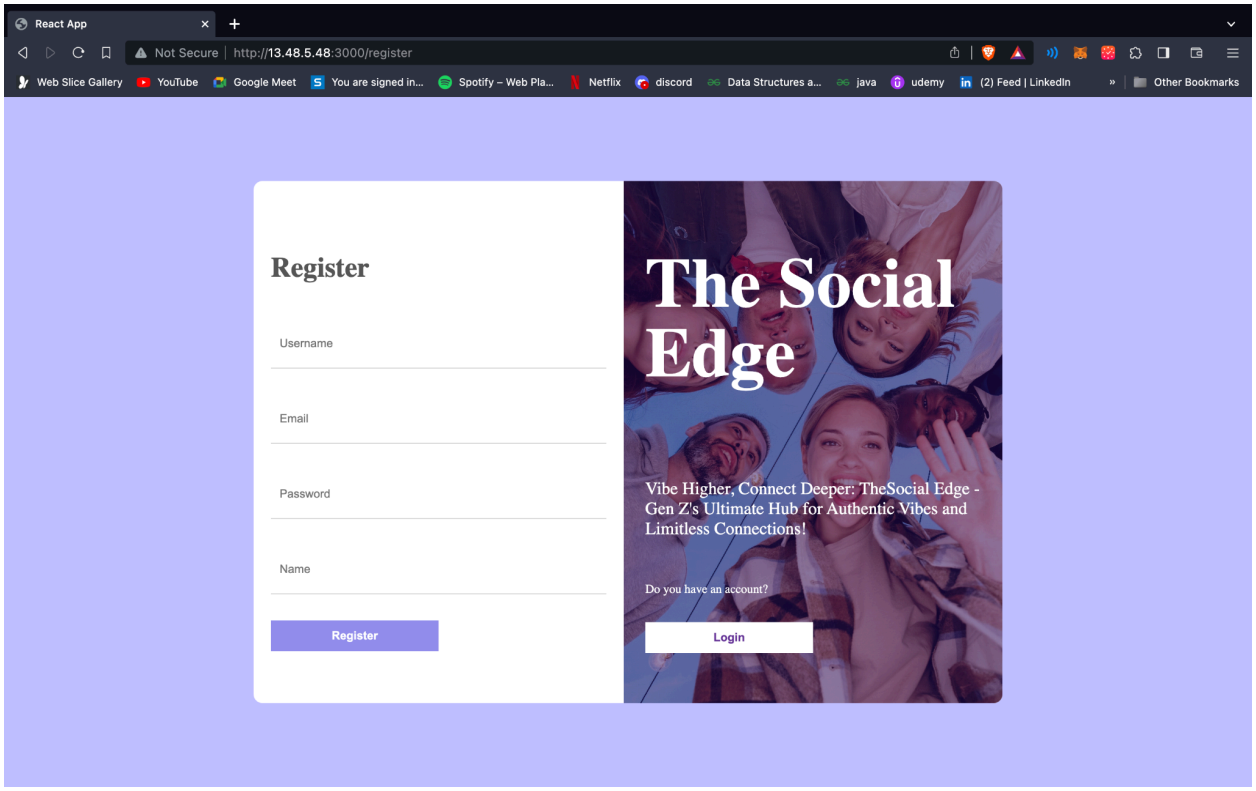
DOCKER CONTAINERS IN RUNNING STATE:

FINAL PROJECT SNAPSHOTS

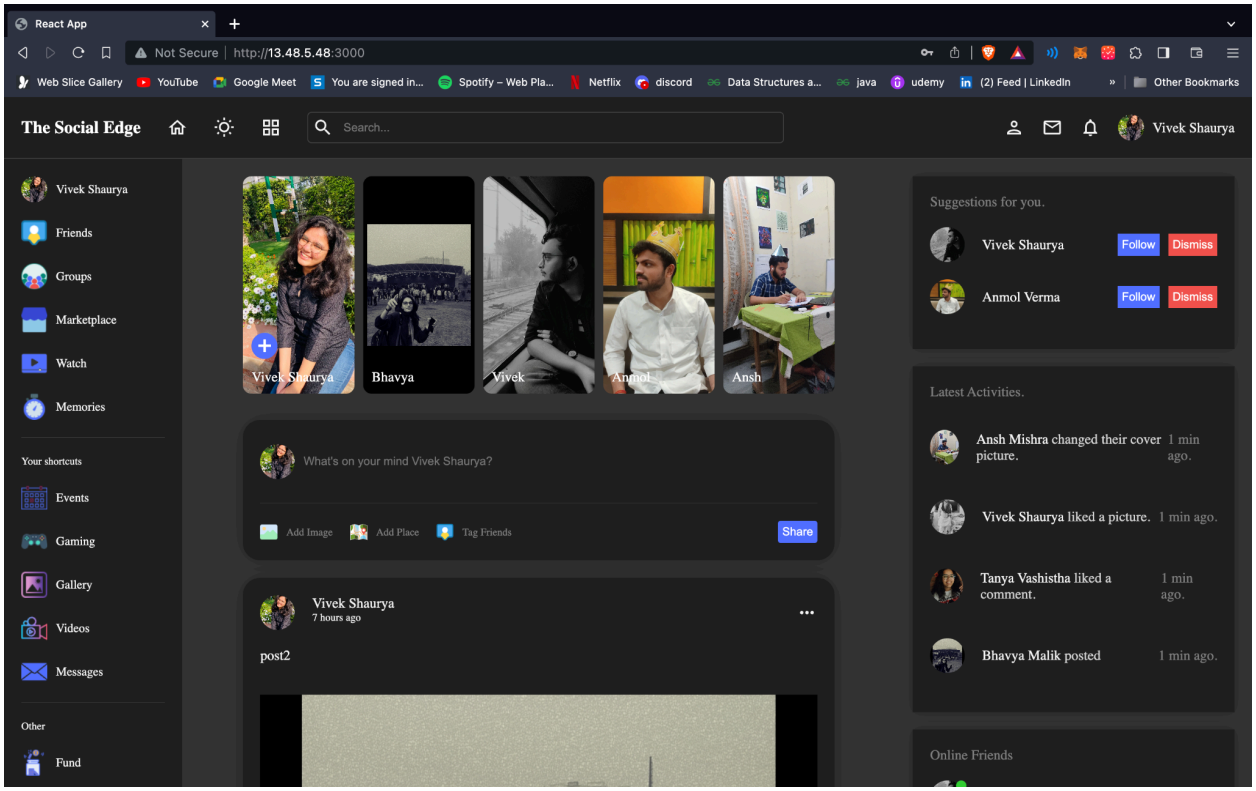
LOGIN PAGE:



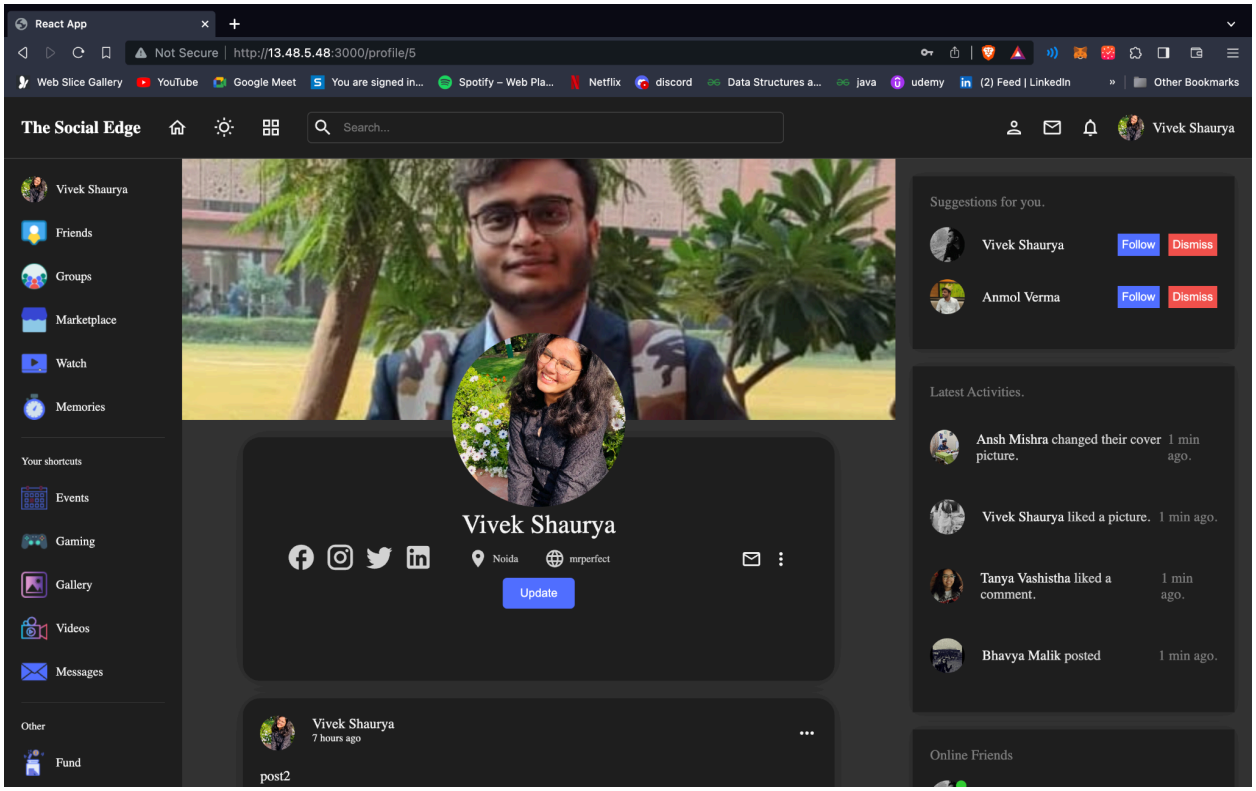
REGISTER PAGE:



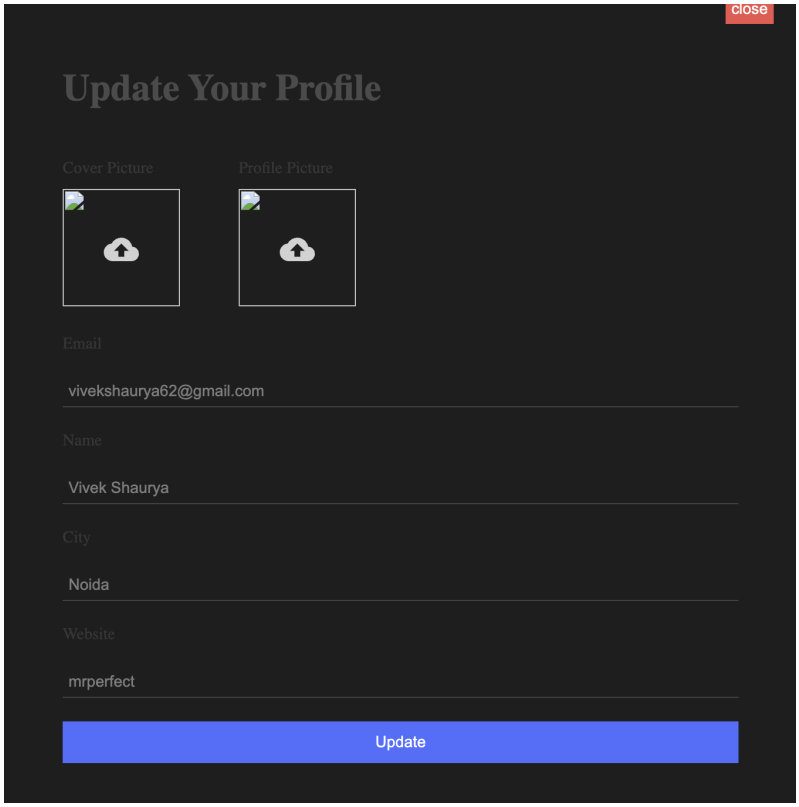
HOME PAGE:



PROFILE PAGE:

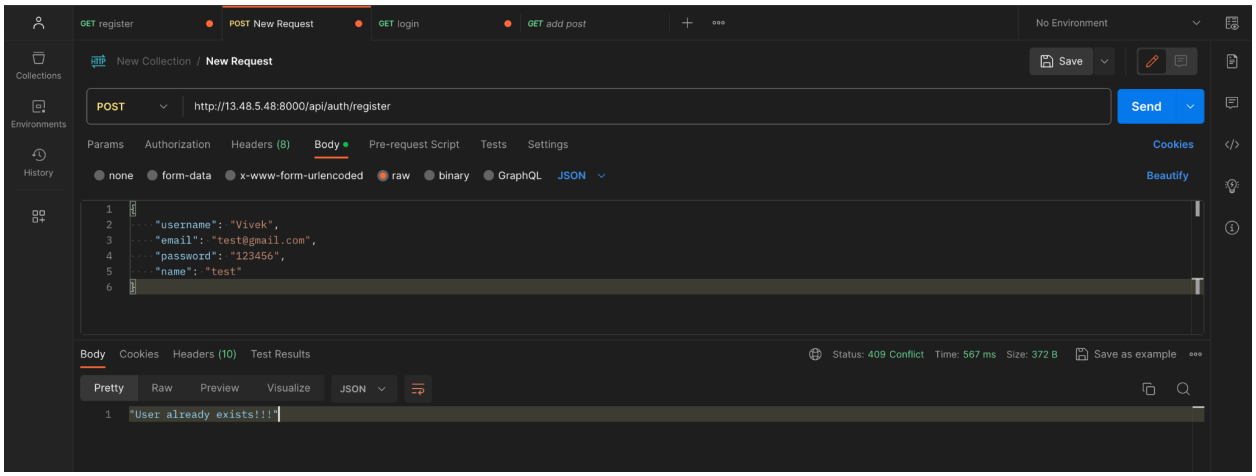


UPDATE PROFILE PAGE:

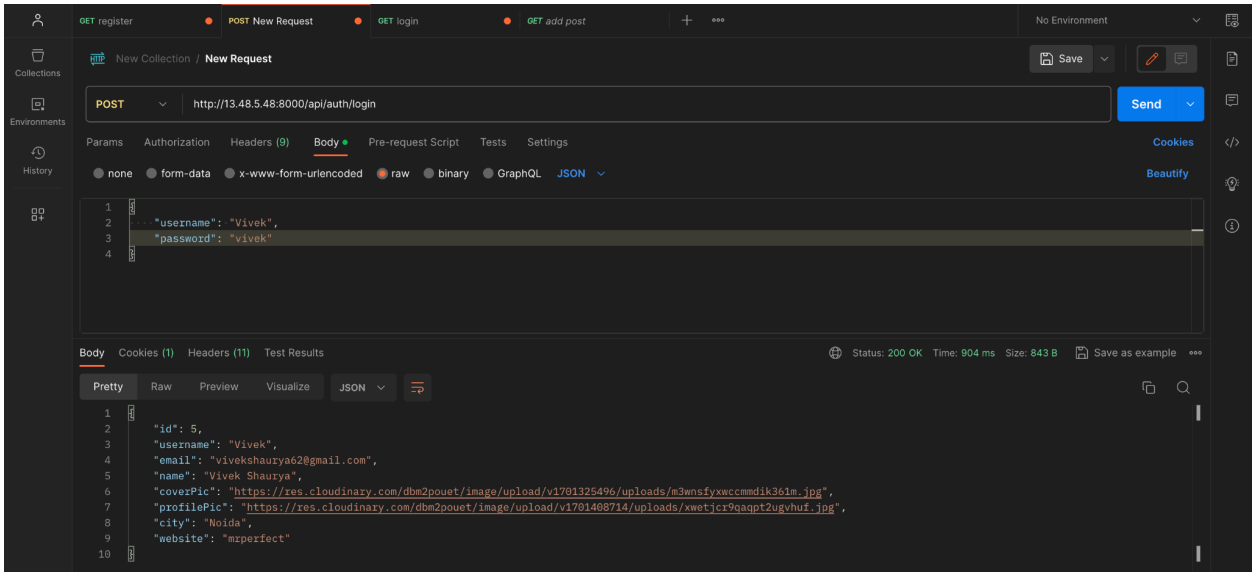


CHAPTER-5
TESTING

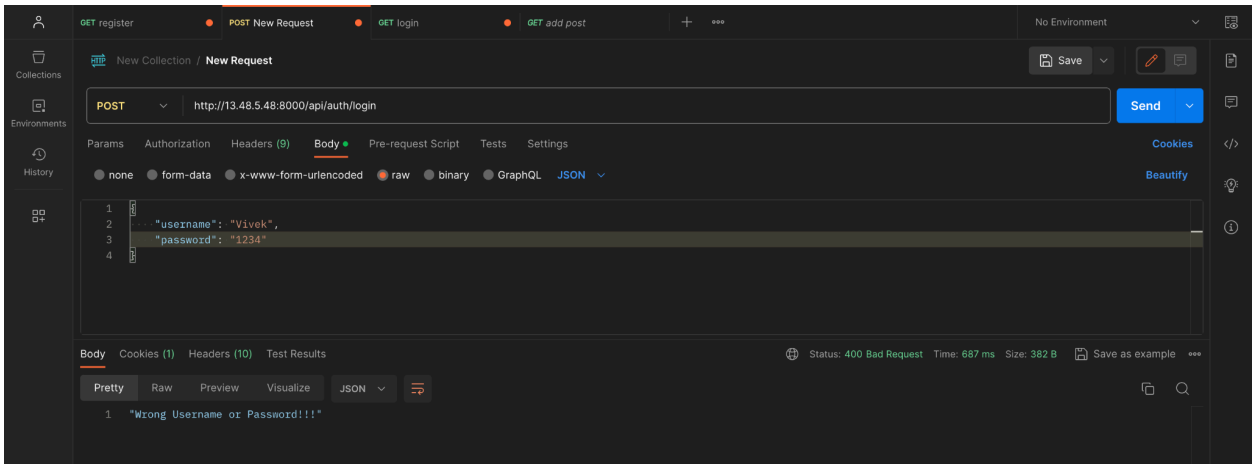
API testing using Postman provides a comprehensive and user-friendly approach to ensure the functionality, reliability, and security of web APIs. Postman simplifies the testing process by offering an intuitive interface for designing, executing, and automating API requests. Users can create collections of requests, organize them logically, and execute them in sequence or parallel. Postman supports various HTTP methods, authentication mechanisms, and request/response types, making it versatile for testing different API scenarios. Additionally, it enables the creation of test scripts using JavaScript, allowing for automated validation of API responses. Postman's features, including environment variables, pre-request scripts, and detailed response visualization, enhance the efficiency and effectiveness of API testing, making it an indispensable tool for developers and QA professionals alike.



API testing for the registration route is conducted using Postman, where a JSON file containing username, email, password, and name is sent to the server. The test assesses proper data transmission and processing, validating the server's response for successful user registration, ensuring the accuracy and security of the registration process.



Conducting API testing with Postman on the login route involves sending a JSON file containing only the username and password to the server. The evaluation ensures secure data transmission and processing, and the server's response, intentionally excluding the password, upholds security integrity, affirming a successful and privacy-conscious authentication process.



CHAPTER-6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In conclusion, our journey to develop a containerized social media website, incorporating Dockerization and CI/CD pipelines, has marked a transformative stride in modern software development. Leveraging Docker for frontend (React), backend (Node.js), and MySQL database components has not only enhanced portability but also ensured consistent performance across diverse environments. The implementation of CI/CD pipelines, orchestrated through GitHub Actions and deployed on AWS EC2 instances, has fortified our development workflow. Automated testing and deployment procedures have significantly reduced errors, ensuring a robust and reliable social media platform.

The synergy between Docker and CI/CD has streamlined collaboration, minimized deployment delays, and fostered an environment where innovation can thrive. The scalability, security, and efficiency achieved through Docker containers showcase the platform's readiness to handle varying levels of user engagement. GitHub Actions, coupled with AWS EC2, have empowered us to seamlessly integrate and deploy updates, responding swiftly to user needs.

The success lies not just in the features of our social media platform but in the commitment to modern development practices. This project stands as a testament to the power of containerization and automation, exemplifying how technological advancements can redefine the landscape of social connectivity in the digital era. The journey continues, marked by adaptability, collaboration, and a commitment to delivering an exceptional user experience in the evolving realm of social media technology.

6.2 FUTURE WORK

The successful completion of the initial phase of our social media project serves as a foundation for ongoing improvements and expansions. As we look ahead, the following areas present opportunities for future work and development:

Advanced Features:
Integrate advanced features such as real-time notifications, multimedia content sharing, and personalized user experiences to enhance user engagement.

Machine Learning Integration:
Explore the integration of machine learning algorithms for personalized content recommendations, sentiment analysis, and user behavior prediction, making the platform more intelligent and user-centric.

Microservices Architecture:
Consider transitioning to a microservices architecture to further modularize and scale individual components independently. This can enhance maintainability and scalability.

User Analytics and Insights:
Implement robust analytics tools to gather insights into user behavior, preferences, and platform usage. This data can inform future feature development and marketing strategies.

Blockchain Integration:
Explore the integration of blockchain technology for enhanced security, transparency, and decentralized content storage, providing users with greater control over their data.

Accessibility Improvements:

Ensure that the platform adheres to accessibility standards, making it inclusive and usable for users with diverse abilities.

User Feedback Mechanism:

Implement a robust user feedback mechanism to collect and analyze user suggestions and concerns, fostering a community-driven approach to continuous improvement.

REFERENCES

1. Amit M Potdar, Narayan D G, Shivaraj Kengond, Mohammed Moin Mulla,Performance Evaluation of Docker Container and Virtual Machine,Procedia Computer Science,Volume 171,2020,Pages 1419-1428,ISSN 1877-0509.
2. D. Reis, B. Piedade, F. F. Correia, J. P. Dias and A. Aguiar, "Developing Docker and Docker-Compose Specifications: A Developers' Survey," in IEEE Access, vol. 10, pp. 2318-2329, 2022, doi: 10.1109/ACCESS.2021.3137671.
3. *Docker compose Overview* (2023) *Docker Documentation*. Available at: https://docs.docker.com/compose/ (Accessed: 01 December 2023).
4. https://medium.com/bb-tutorials-and-thoughts/how-to-build-and-deploy-mern-stack-on-aws-ecs-2a760a1f4de1 (Accessed on 01-11-2023)
5. https://tanstack.com/query/latest/ (Accessed on 19-10-2023)
6. https://nodejs.org/en/docs/guides/nodejs-docker-webapp (Accessed on 05-11-2023)
7. https://github.com/hajar-elkhalidi/docker-compose-github-actions (Accessed on 18-10-2023)
8. https://baillahiamine.medium.com/dockerize-a-mern-stack-app-bd3c9fd2e1fb (Accessed on 22-10-2023)
9. https://medium.com/@sudhirmeena/getting-started-with-node-docker-and-mysql-aff3cd7b8395 (Accessed on 01-11-2023)
10. https://medium.com/codex/adding-authentication-to-full-stack-mern-web-application-4974543c3e66 (Accessed on 01-10-2023)
11. https://medium.com/dont-code-me-on-that/recap-jwt-and-cookies-for-web-authn-80394de8e306(Accessed on 01-10-2023)