Dockerized Entertainment React Service

Course Name: DevOps Foundation

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1	Yuvraj Jaiswal	EN22CS303060
2	Rohit Marmat	EN22CS303045
3	Bhumika Gwala	EN22CS304021
4	Suyash Jain	EN22CS3011005
5	Divyansh Parmar	EN22CS306017
6	Aayush Yadav	EN22CS303002

Group Name: Group 08D11

Project Number: DO-34

Industry Mentor Name: Mr. Vaibhav

University Mentor Name: Prof. Shyam Patel

Academic Year: 2025-2026

1. **Problem Statement & Objectives**
    1. Problem Statement
    2. Project Objectives
    3. Scope of the Project
2. **Proposed Solution**
    1. Key features
    2. Overall Architecture / Workflow
    3. Tools & Technologies Used
3. **Results & Output**
    1. Screenshots / outputs
    2. Reports / dashboards / models
    3. Key outcomes
4. **Conclusion**
5. **Future Scope & Enhancements**

Problem Statement & Objectives

1. Problem Statement

Modern web applications often suffer from inconsistent development environments, deployment challenges, and integration issues between frontend, backend, and database components. Manual deployments lead to configuration mismatches, downtime, and scaling difficulties.

The objective of this project is to containerize an Entertainment Service application built using React and FastAPI using Docker best practices. The project also aims to automate deployment using CI/CD pipelines and ensure portability across development and production environments.

---

2. Project Objectives

1. Containerize React frontend using multi-stage Docker build.
2. Containerize FastAPI backend with optimized Python image.
3. Integrate MySQL database container.
4. Create Docker Compose configuration for service orchestration.
5. Implement CI/CD pipeline using GitHub Actions.
6. Deploy the application on AWS EC2.
7. Ensure consistent, portable, and scalable deployment architecture.

---

3. Scope of the Project

The project focuses on:

- Containerization of frontend, backend, and database services.
- Image optimization using multi-stage builds.
- Service orchestration using Docker Compose.
- Automated deployment pipeline using GitHub Actions.
- Cloud deployment on AWS EC2.

Proposed Solution

The solution involves creating isolated Docker containers for each service:

- React frontend (served via Nginx)
- FastAPI backend (served via Uvicorn)
- MySQL database

A Docker Compose file orchestrates these services and handles networking and dependencies.

CI/CD automation is implemented using GitHub Actions. On every push to the main branch, Docker images are built and pushed to Docker Hub. The pipeline then connects to an EC2 instance and redeploys the updated containers automatically.

---

1. Key Features

- Multi-stage Dockerfile for frontend optimization
- Lightweight Python backend container
- MySQL container with persistent volume
- Docker Compose orchestration
- REST API with authentication
- CI/CD pipeline using GitHub Actions
- Automated cloud deployment on AWS EC2
- Secure environment variable configuration
- Service-to-service communication using Docker networking

---

2. Overall Architecture / Workflow

2.1 Development Workflow

Developer

↓

GitHub

↓

GitHub Actions

↓

Docker Hub

↓

AWS EC2

↓

Docker Compose

↓

Running Containers

2.2 Application Architecture

Frontend (React + Nginx)

↓

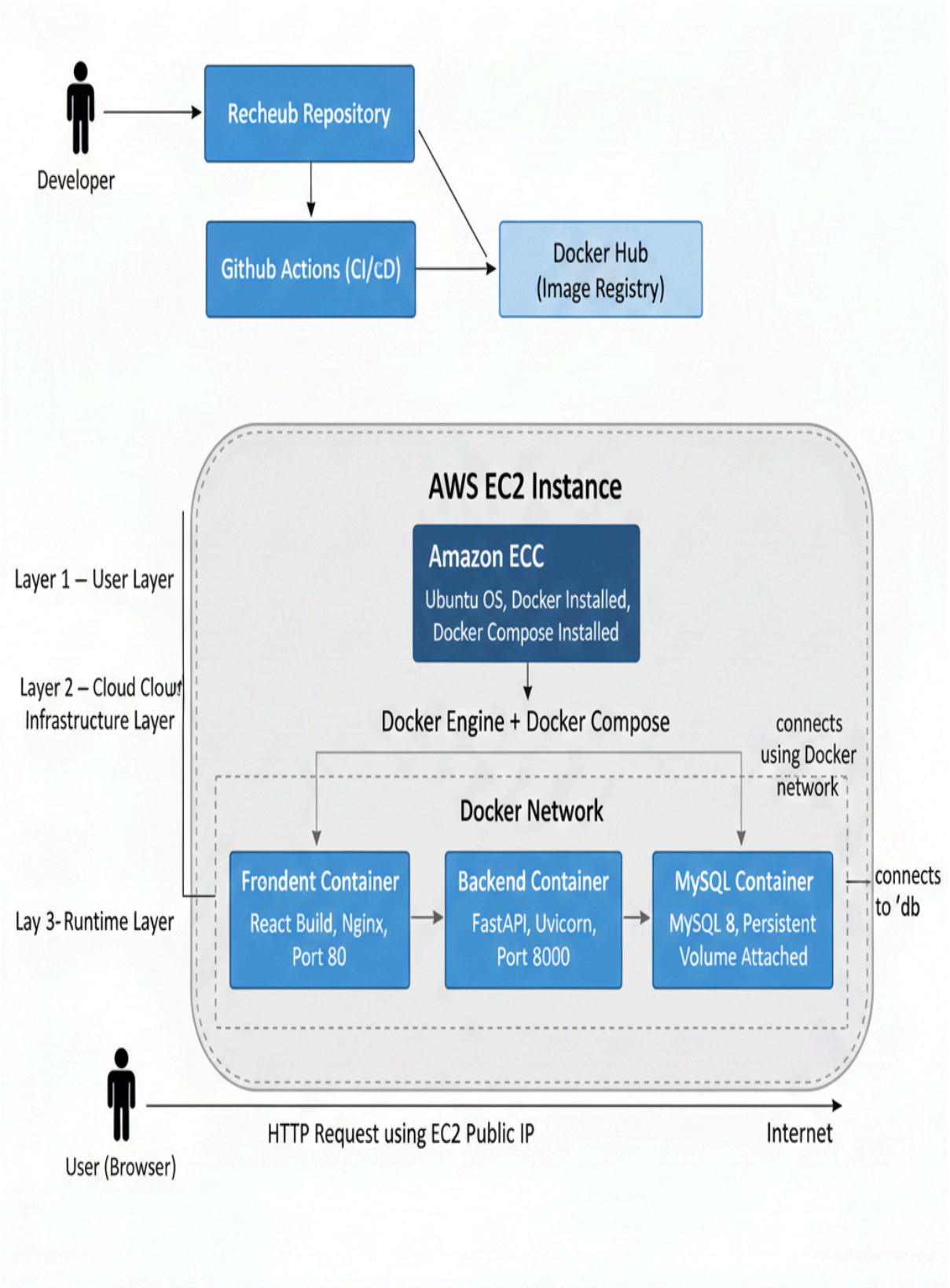
Backend (FastAPI + Uvicorn)

↓

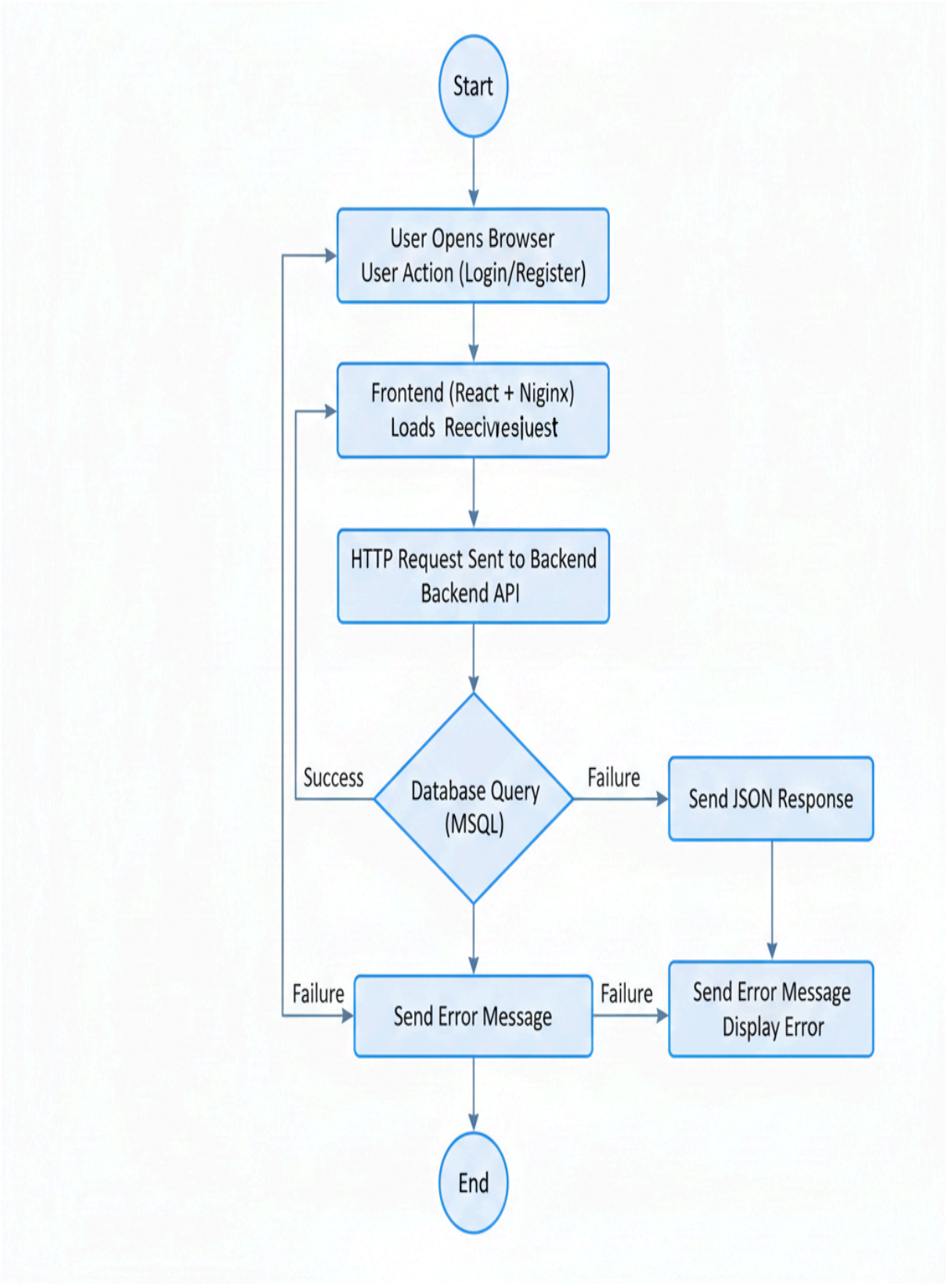
MySQL Database

All services communicate over a Docker network.

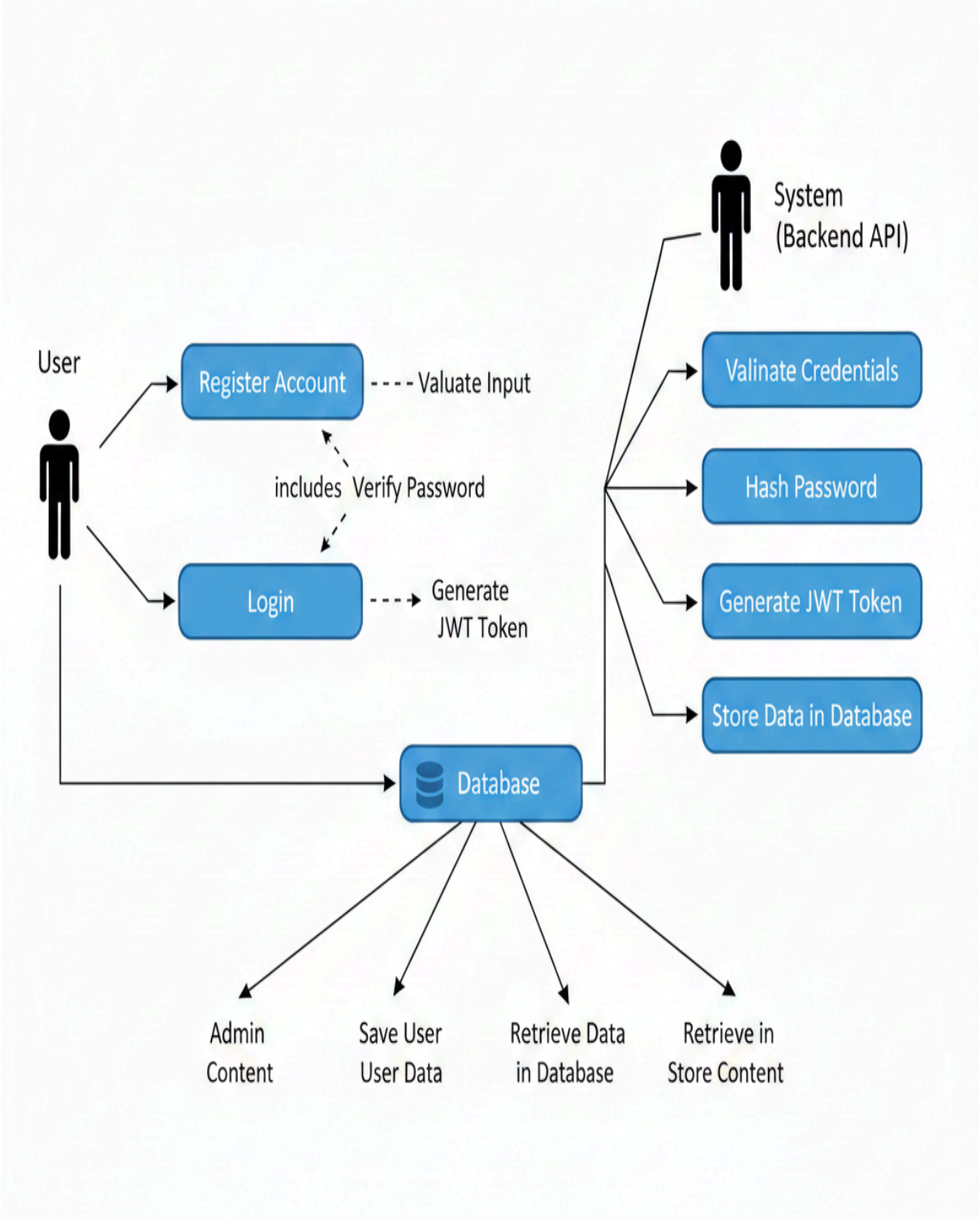
2.3 System Architecture



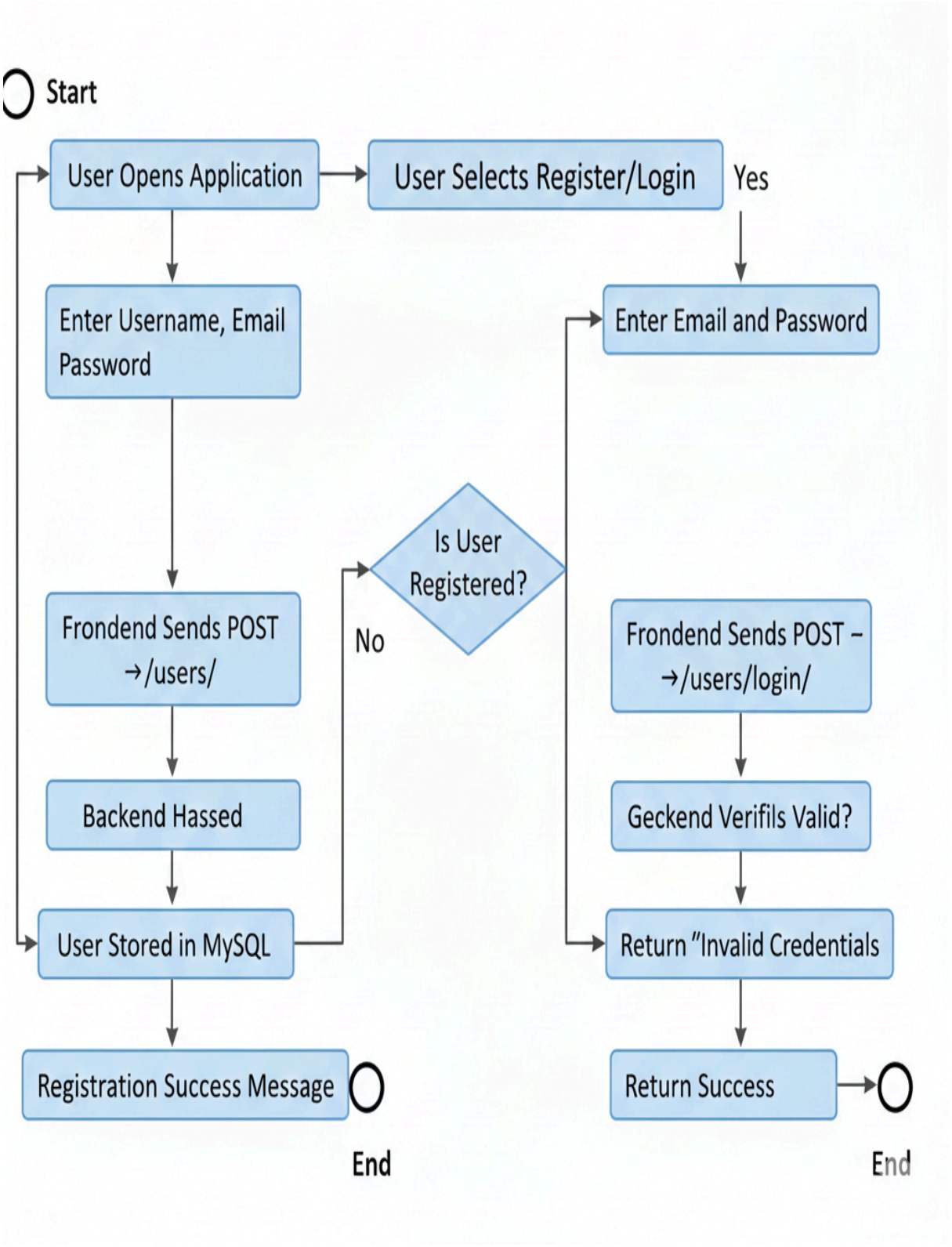
2.4 Flow chart



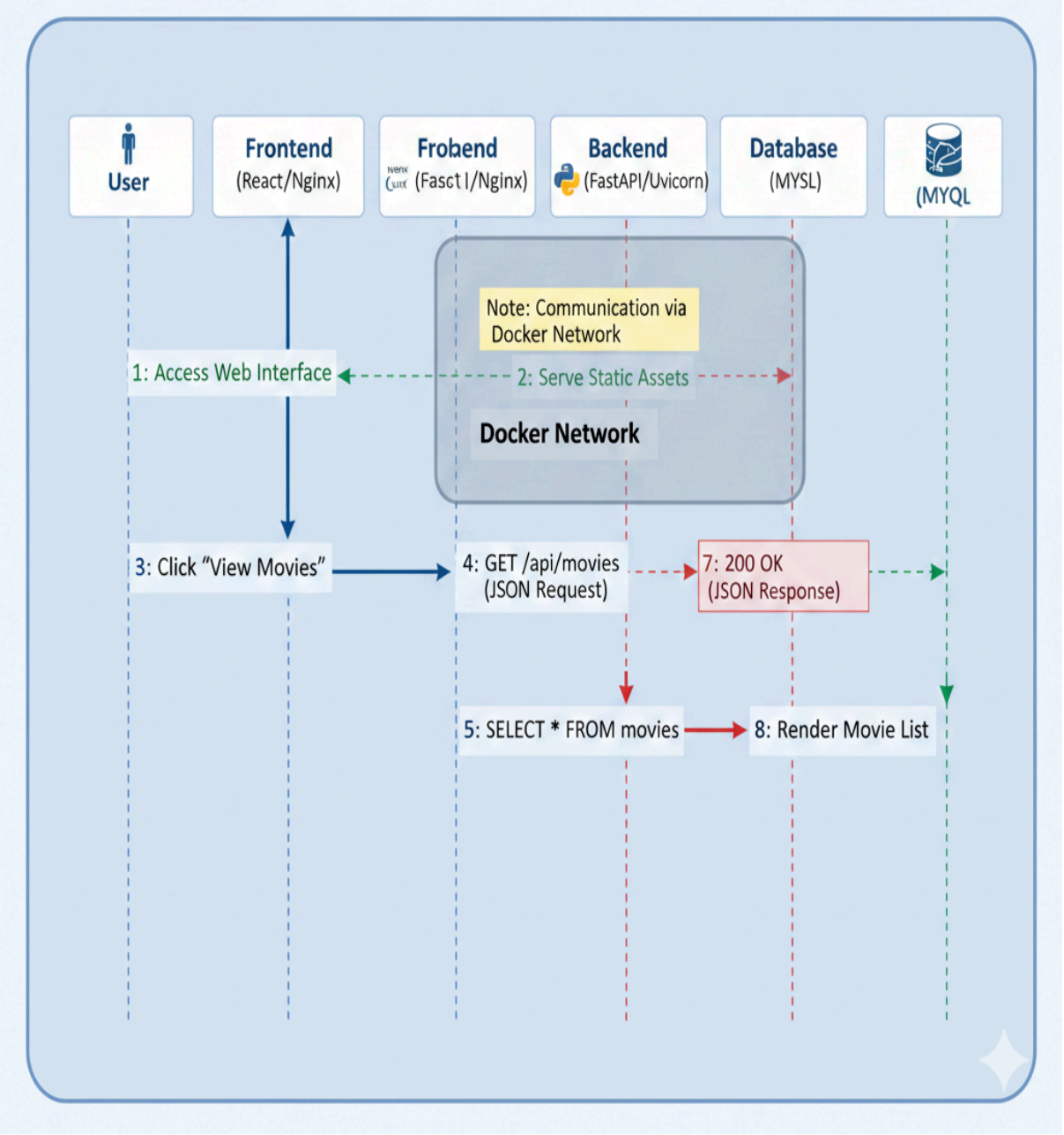
2.5 Use case diagram



2.6 Activity Diagram



2.7 Sequence diagram



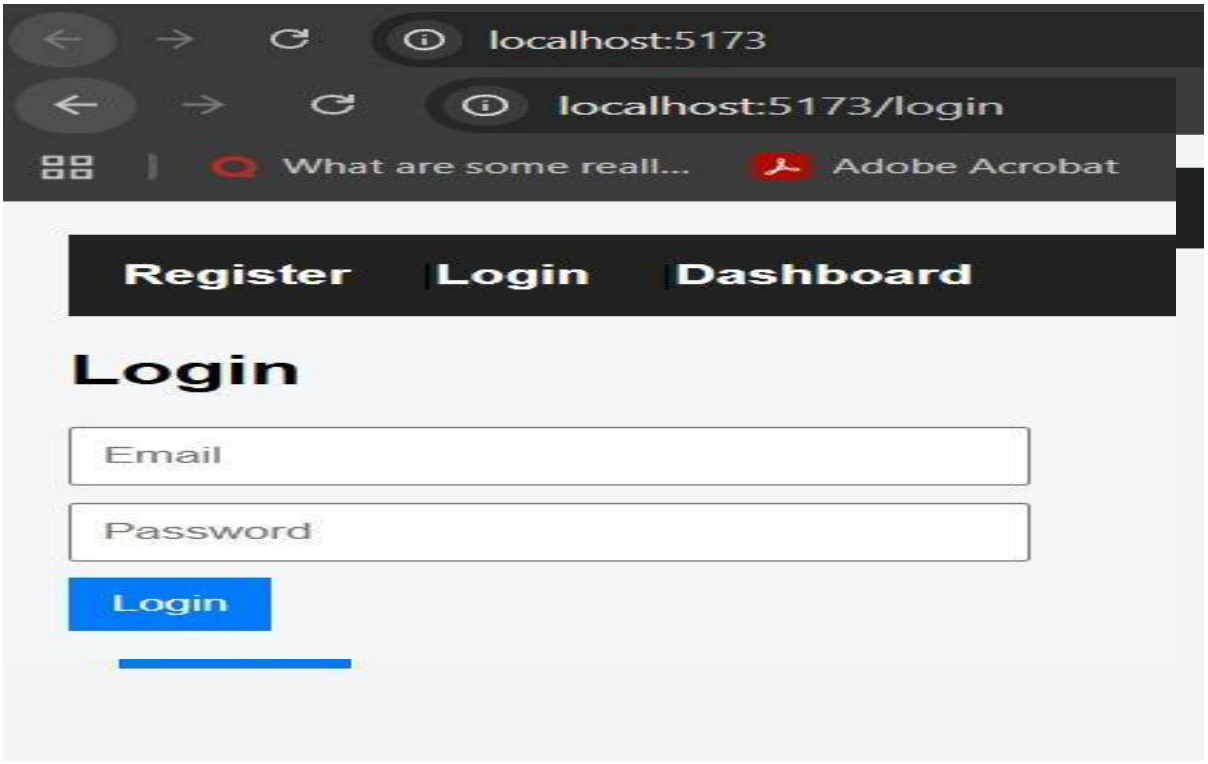
2. Tools & Technologies Used

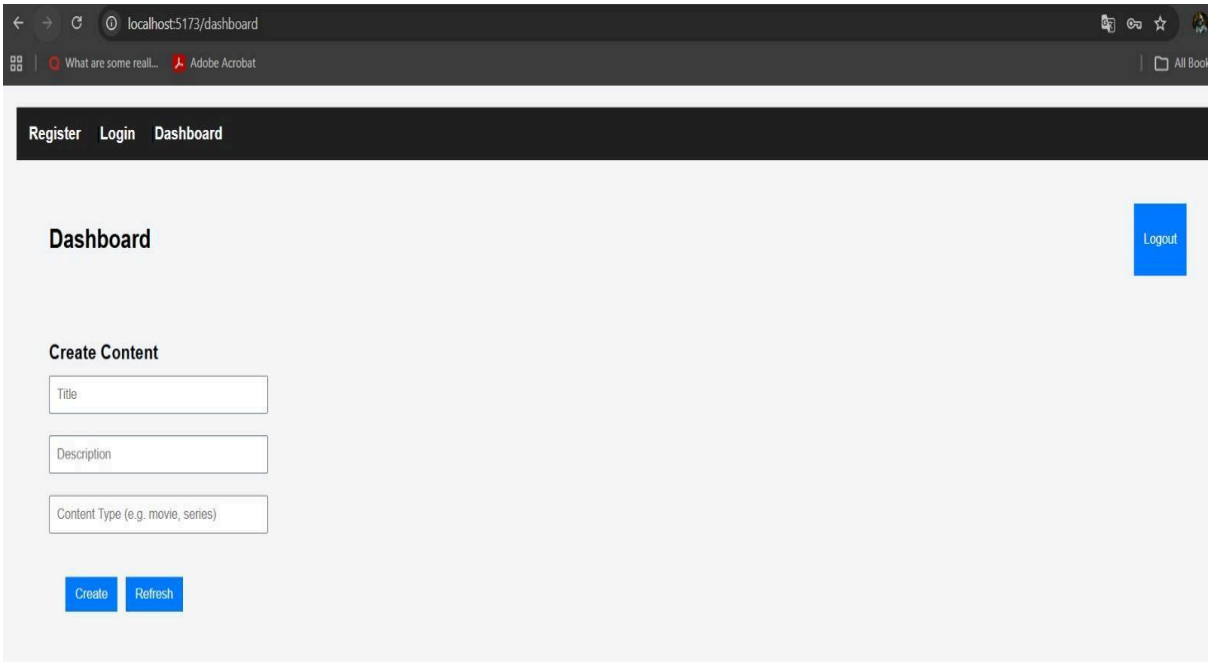
- Git (Version Control)
- Docker (Containerization)
- Docker Compose (Service Orchestration)
- GitHub Actions (CI/CD)
- AWS EC2 (Cloud Deployment)
- React (Frontend)
- FastAPI (Backend)
- MySQL (Database)
- Nginx (Frontend Serving)
- Uvicorn (ASGI Server)

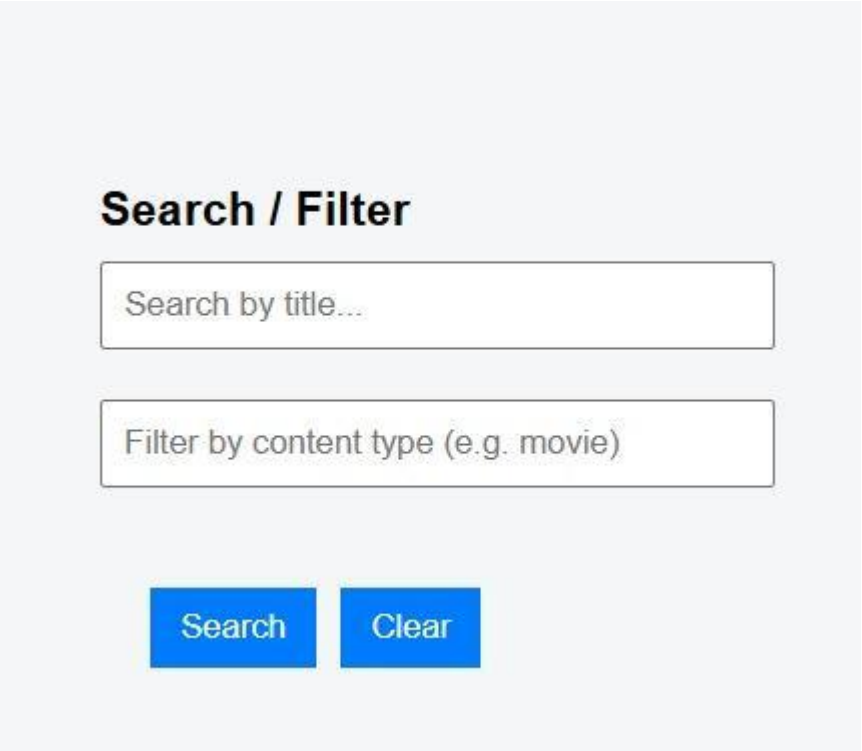
Results & Output

1. Screenshots / Outputs

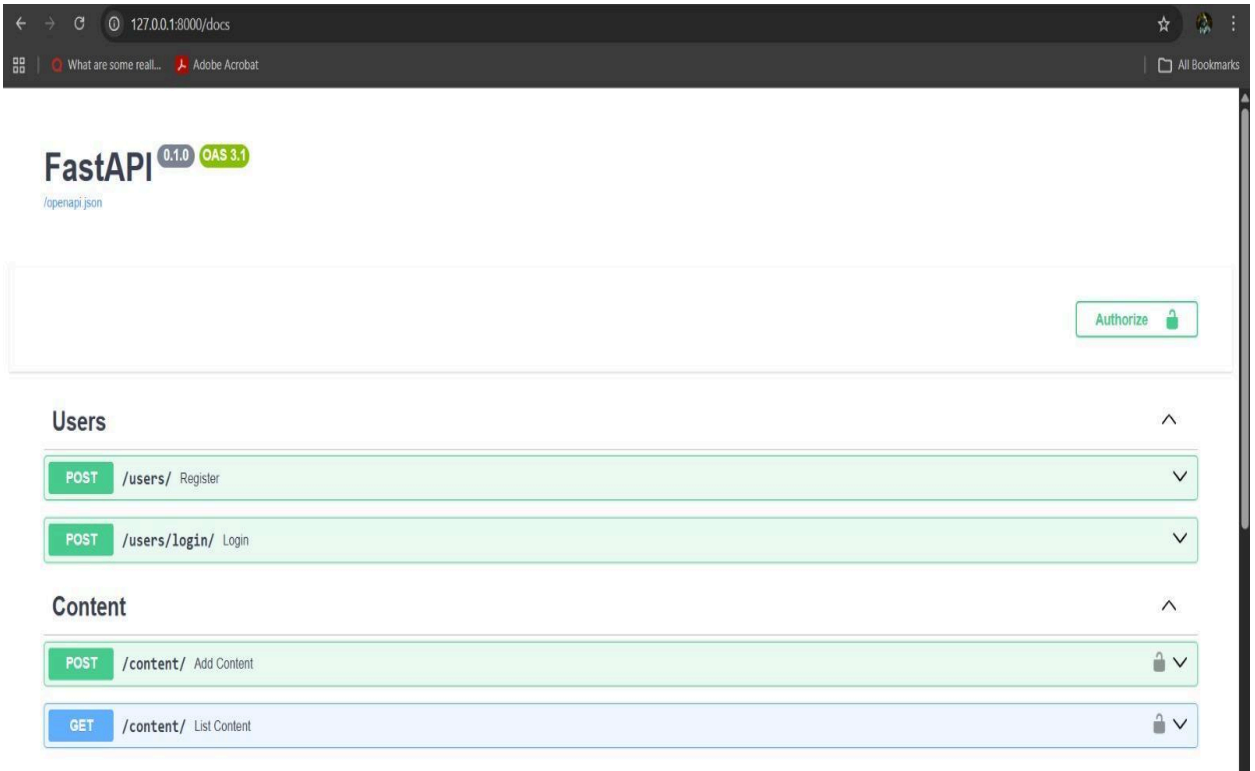
• Login/Register UI



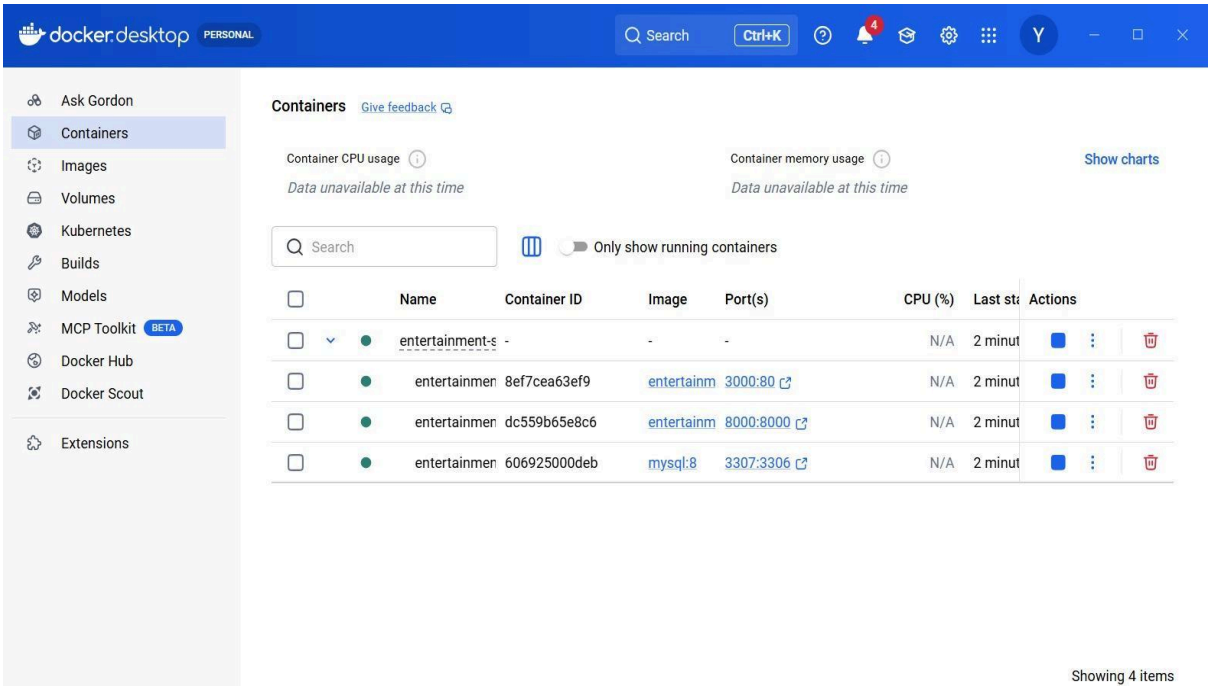




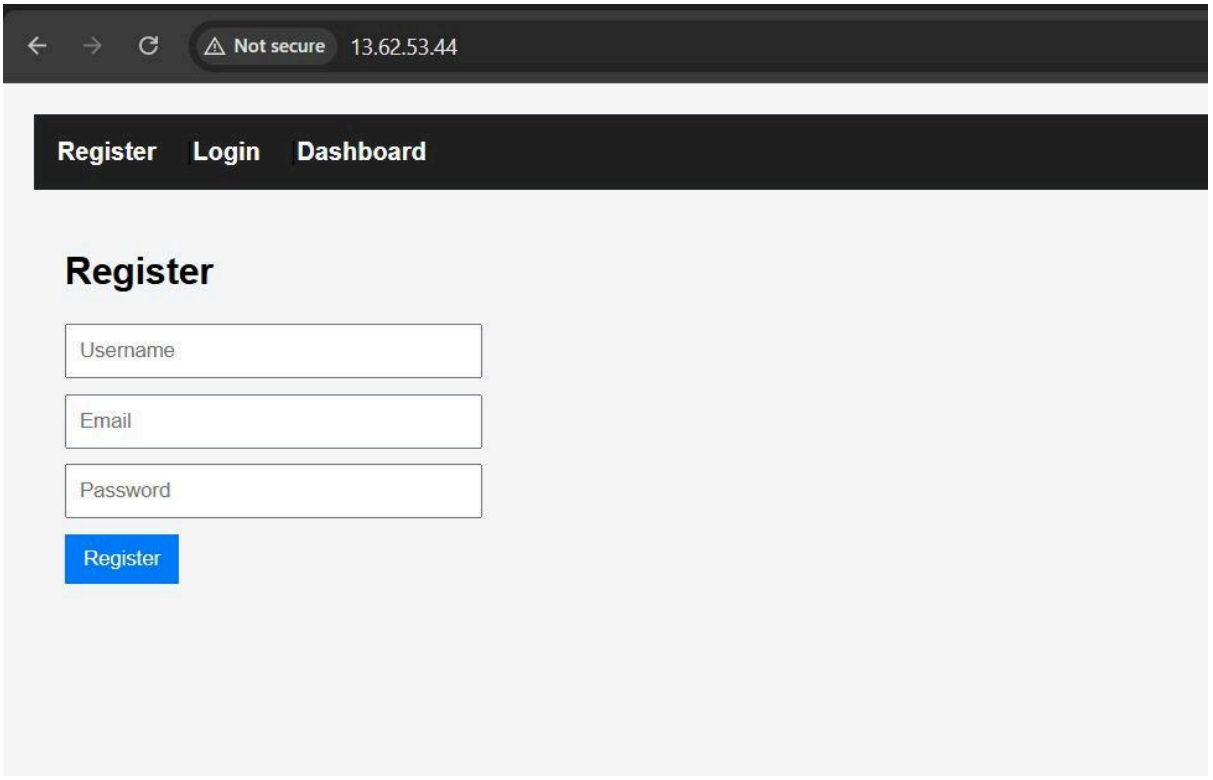
- Swagger API documentation



• Running containers (docker ps)



- EC2 deployment



• GitHub Actions successful run

2. Reports / Dashboards / Models

• CI/CD execution logs

• Docker image build logs

• Container health status

3. Key Outcomes

- Successfully containerized multi-service application.
- Reduced environment dependency conflicts.
- Automated build and deployment pipeline.
- Achieved consistent production-ready deployment.
- Improved scalability and maintainability.

Conclusion

The Dockerized Entertainment React Service project successfully demonstrates the practical implementation of containerization and CI/CD automation using modern DevOps practices. By leveraging Docker and GitHub Actions, the application ensures consistent development and deployment environments. Cloud deployment on AWS EC2 further enhances portability and real-world applicability.

The project provided hands-on experience in container orchestration, image optimization, backend integration, and automated deployment strategies aligned with industry standards.

Future Scope & Enhancements

1. Deployment using Kubernetes for orchestration.
2. Implementation of HTTPS using SSL certificates.
3. Integration of monitoring tools (Prometheus & Grafana).
4. Add centralized logging (ELK Stack).
5. Implement role-based authentication.
6. Use Docker Secrets for enhanced security.
7. Auto-scaling using Kubernetes or AWS ECS.
8. Infrastructure as Code using Terraform.