Project - Report

College Management System

Course Name: DevOps Foundation

Institution Name: Medicaps University – Datagami Skill Based Course

Sr No	Student Name	Enrolment Number
01	Sneha Jain	EN22CS301966
02	Soumya Barve	EN22CS301975
03	Sumit Pawar	EN22CS301100
04	Vikas Devnani	EN22CS3011080
05	Vikas Dhakad	EN22CS3011081

Group Name: Group 09D9

Project Number: DO-09

Industry Mentor Name:

Mr. Vaibhav University Mentor

Name: Prof. Ritesh Joshi

Academic Year: 2026

Problem Statement & Objectives

1. **Problem Statement** –
   CI/CD for React.js College Management System app

2. **Project Objectives** –

- To design and develop a modular full-stack College Management System supporting Admin, Faculty, and Student roles.
- To implement secure authentication and role-based authorization using JWT.
- To design a normalized relational database (3NF) ensuring data integrity and consistency.
- To containerize the application using Docker for environment consistency.
- To implement a CI/CD pipeline using GitHub Actions for automated build, testing, and deployment.
- To deploy the system on AWS EC2 using Kubernetes (k3s) for orchestration and scalability.
- To ensure high availability, security, and automated rollback mechanisms.
- To demonstrate an enterprise-level DevOps workflow integrated with application architecture.

3. **Scope of the Project**-

- Full-stack web-based College Management System
- Student, Faculty, and Admin modules
- JWT-based authentication and RBAC
- Relational database schema design
- Docker containerization
- CI/CD automation using GitHub Actions
- Deployment on AWS EC2
- Kubernetes (k3s) orchestration
- Persistent volume configuration
- Secure secret management
- Automated deployment pipeline

Proposed Solution

The proposed solution is a **DevOps-enabled Full Stack College Management System** that integrates application development with infrastructure automation and container orchestration.

The system follows a layered architecture and uses CI/CD automation to transition from code commit to production deployment with minimal manual intervention.

1. Key features -

- Role-Based Access Control (Admin, Faculty, Student)
- Secure JWT Authentication
- Modular RESTful API Architecture
- Normalized Relational Database Design
- Docker Multi-Stage Containerization
- GitHub Actions CI/CD Automation
- Kubernetes (k3s) Orchestration
- Self-Healing Pods
- Rolling Updates with Zero Downtime
- Secure Secret Management
- Persistent Data Storage using Kubernetes Volumes
- Automated Build & Deployment Pipeline

2. Overall Architecture / Workflow

The CI/CD workflow follows a structured multi-stage process:

Stage 1 – Source Code Management

- Developer pushes code to Git repository.
- GitHub (or Jenkins) detects the push event.
- CI pipeline is automatically triggered.

Stage 2 – Continuous Integration

- Dependencies are installed.
- Automated unit tests are executed.
- React production build is generated.

If any test fails:

- Pipeline execution stops.
- Deployment is prevented.

If all tests pass:

- Pipeline proceeds to containerization stage.

Stage 3 – Containerization

- Docker image is built using a multi-stage Dockerfile.
- Optimized production image is generated.
- Image is tagged using version or commit hash.
- Image is pushed to Docker Hub (or private registry).

This ensures:

- Immutable artifact creation
- Environment consistency
- Easy rollback using version tags
-

Stage 4 – Continuous Deployment

Upon merging code into the production branch:

- Deployment workflow is triggered.
- Kubernetes pulls the latest container image.
- Rolling update process begins.
- Old pods are gradually replaced with new pods.
- Zero downtime is maintained.

Stage 5 – Production Environment

- React CMS app is exposed via Kubernetes Service (NodePort/ClusterIP).
- Pods are monitored by Kubernetes.
- Failed pods are automatically restarted (self-healing).
- Application becomes accessible to end users.

3. Tools & Technologies Used-

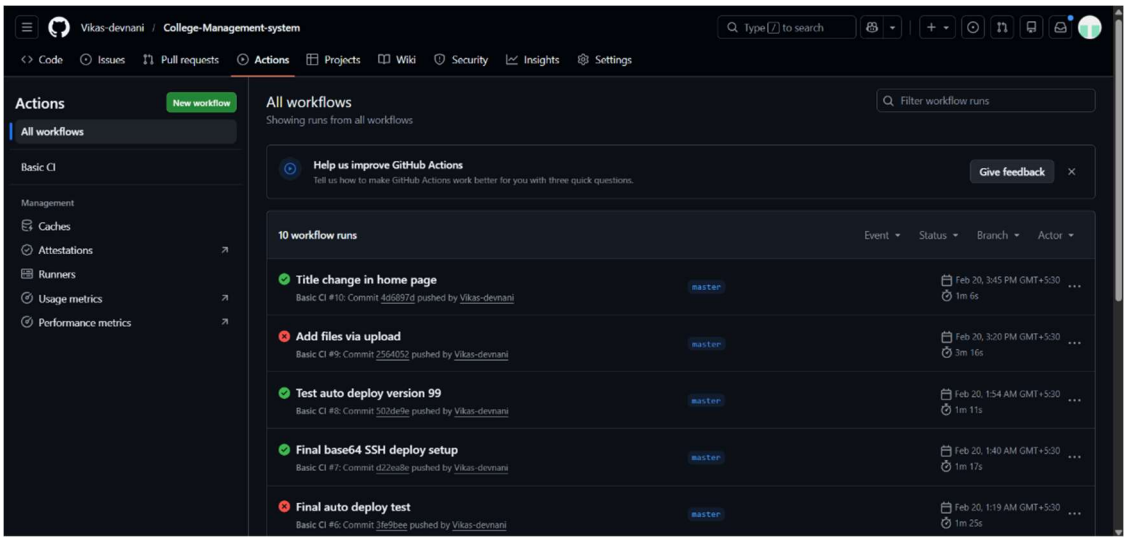
Category	Tools Used
Version Control	Git
Repository Hosting	GitHub
Frontend Framework	React.js
CI/CD Engine	GitHub Actions / Jenkins
Containerization	Docker
Container Registry	Docker Hub
Orchestration	Kubernetes (K8s)
Deployment Strategy	Rolling Update
Infrastructure	Cloud VM (AWS EC2)

Results & Output

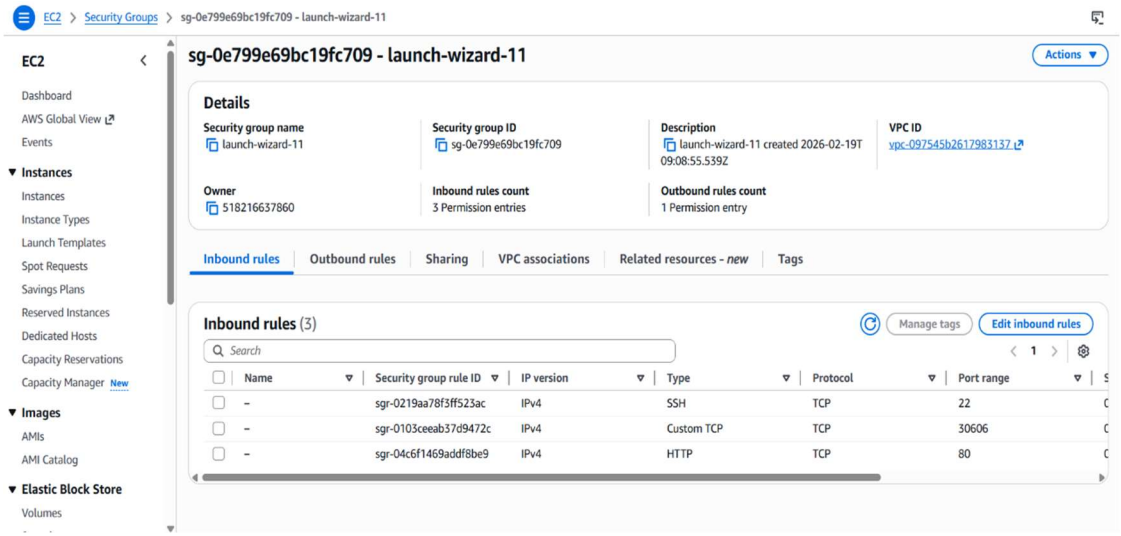
1. Screenshots / Outputs-

The following outputs were generated during project execution:

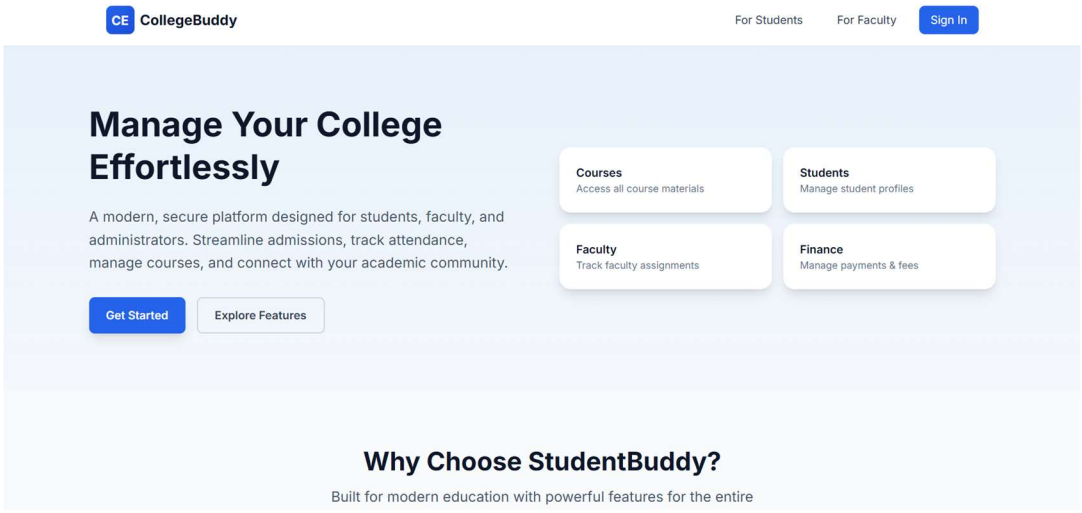
- Git commit triggering CI workflow

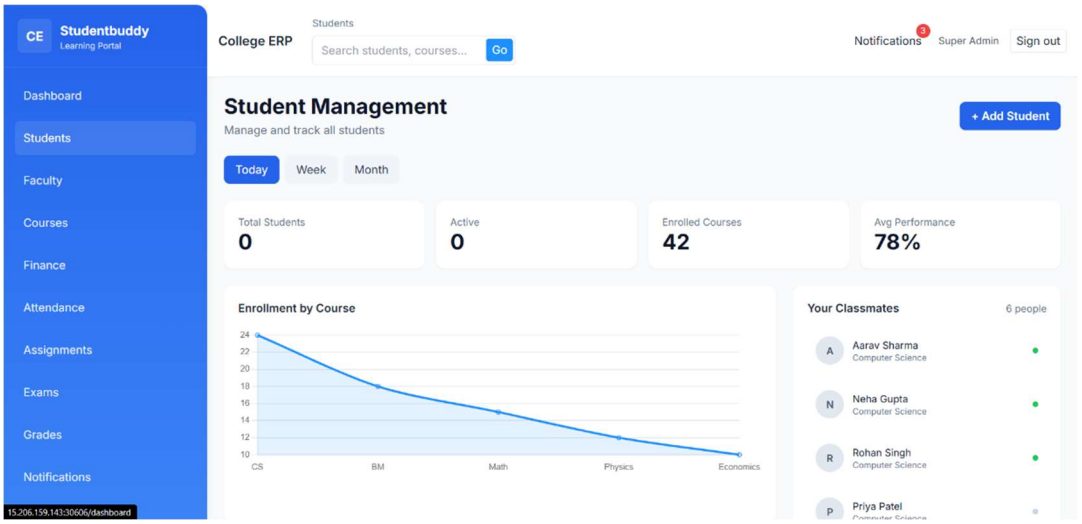


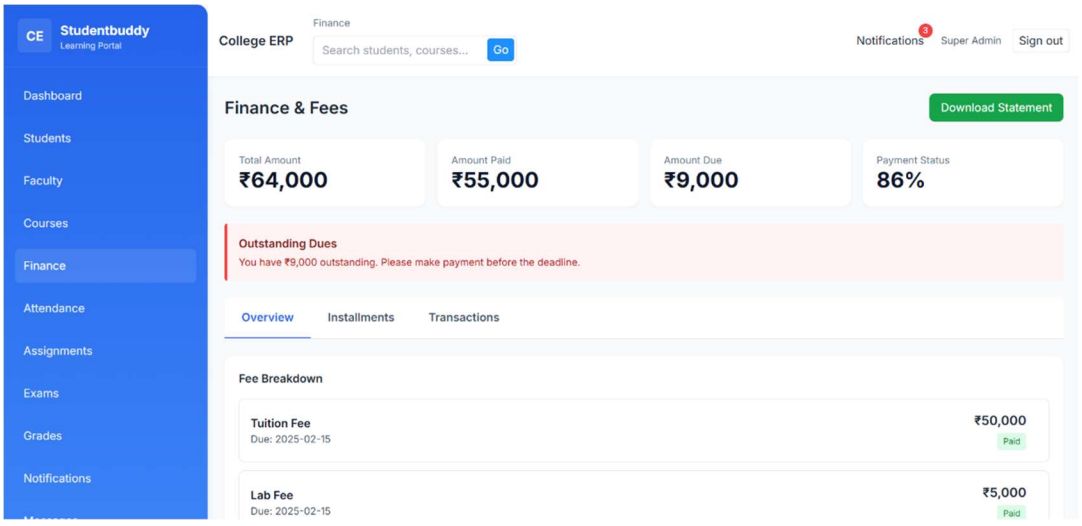
- AWS EC2 Security Group Inbound Rules Configuration



- Live React CMS application in browser







3. Key Outcomes

- Successfully automated build and deployment lifecycle.
- Reduced deployment time significantly.
- Eliminated manual deployment steps.
- Ensured consistent production environments.
- Achieved automated test validation on every commit.
- Implemented version-controlled container artifacts.
- Enabled zero-downtime rolling deployments.
- Demonstrated scalable Kubernetes-based architecture.
- Achieved enterprise-level CI/CD automation for the College Management System.

Conclusion

The project successfully implemented a professional CI/CD pipeline for the React.js College Management System application using Git, Docker, GitHub Actions, and Kubernetes.

By automating testing, build, containerization, and deployment processes, the system ensures:

- High code quality
- Faster release cycles
- Reliable and repeatable deployments
- Reduced human errors
- Scalable enterprise-grade infrastructure

This project provided hands-on experience in implementing modern DevOps practices and demonstrated how automation enhances reliability, efficiency, and scalability in real-world applications.

Future Scope & Enhancements

- Integration of Prometheus and Grafana for monitoring
- Horizontal Pod Autoscaler (HPA)
- Blue-Green deployment strategy
- Canary deployments
- Automated vulnerability scanning in CI pipeline
- Advanced logging using ELK stack
- Multi-environment staging pipelines
- Infrastructure as Code using Terraform
- Advanced security hardening