

# Continuous Integration and Delivery Pipeline

Sprint Number 3

Date 11/3/2024

Name	Email Address
John Gahagan	john.gahagan398@topper.wku.edu
Jay Mistry	jay.mistry627@topper.wku.edu

CS 396

Fall 2024

Project Technical Documentation

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project Overview . . . . .	1
1.2	Project Scope . . . . .	1
1.3	Technical Requirements . . . . .	1
1.3.1	Functional Requirements . . . . .	1
1.3.2	Non-Functional Requirements . . . . .	2
<b>2</b>	<b>DevOps - Continuous Integration and Continuous Delivery Approach and Results</b>	<b>2</b>
<b>3</b>	<b>DevOps - Architecture Approach, Models, and Results</b>	<b>3</b>
<b>4</b>	<b>DevOps - Product and Process Approach and Results</b>	<b>5</b>
<b>5</b>	<b>DevOps - Product Management and Monitoring Approach and Results</b>	<b>5</b>
<b>6</b>	<b>DevOps - Cultural Approach and Results</b>	<b>5</b>
<b>7</b>	<b>Software Testing and Results</b>	<b>6</b>
7.1	Software Testing Plan Template . . . . .	6
<b>8</b>	<b>Conclusion</b>	<b>9</b>
<b>9</b>	<b>Appendix</b>	<b>9</b>
9.1	Software Product Build Instructions . . . . .	9
9.1.1	CI/CD Configuration . . . . .	9
9.2	Software Product User Guide . . . . .	10
9.3	Source Code with Comments . . . . .	10

List of Figures

1	GitHub Actions CI/CD Pipeline Workflow Runs - Showing successful and failed attempts during pipeline configuration . . . . .	3
2	Architecture Diagram Illustrating CI/CD Pipeline, Back end, Front end, and Docker Container Interactions . . . . .	4
3	Example of Alerts and Notifications Setup for Proactive Issue Detection . . . . .	6
4	Failed Login Attempt: Displaying error message for incorrect credentials . . . . .	7
5	Error Handling for Data Fetching Failure: Displaying error message when unable to retrieve patient data . . . . .	7
6	Pipeline Build Times for Each Run Compared to 10-Minute Threshold . . . . .	8

# 1 Introduction

## 1.1 Project Overview

This project aims to design and implement a Continuous Integration and Continuous Deployment (CI/CD) pipeline for a healthcare information system. The main goal is to automate the software delivery process, improving patient data management by enabling efficient and reliable deployments of updates to the system. The system integrates multiple tools and technologies like Node for the back end, React for the front end, Docker for containerization, and GitHub Actions for the CI/CD pipeline. By automating testing, deployment, and monitoring, the system minimizes manual interventions, resulting in improved software quality, faster delivery, and increased stability. The target audience for this system includes healthcare providers, IT administrators, and development teams. By improving the efficiency of software delivery, the healthcare information system can be updated frequently without disrupting critical operations, providing timely features and fixes.

## 1.2 Project Scope

The primary deliverables for this project include a fully functional CI/CD pipeline, back end and front end applications, and the supporting technical documentation. The CI/CD pipeline is designed to build, test, and deploy the healthcare information system automatically whenever new code is pushed to the repository.

## 1.3 Technical Requirements

### 1.3.1 Functional Requirements

Mandatory Functional Requirements
1. The CI/CD pipeline must integrate with popular version control systems (e.g., Git) to automatically trigger builds and deployments when code changes are pushed to the repository.
2. The system must support a configurable automated build process that compiles the healthcare application, generates artifacts, and packages them for deployment, ensuring consistency across environments.
3. The pipeline must include an automated testing framework that executes unit tests, integration tests, and end-to-end tests as part of the build process to validate code quality before deployment.
4. The CI/CD pipeline must implement continuous monitoring and logging of the deployment process, providing real-time feedback on build status, test results, and deployment success or failure to stakeholders.
5. The system must provide a rollback mechanism to revert to the previous version of the application in case of deployment failures, ensuring minimal downtime and maintaining system stability.
Extended Functional Requirements
1. Support for real-time performance monitoring during deployment and operation.
2. Detailed error logging for all stages of deployment to assist in troubleshooting.
3. Ability to manage multiple environments with environment specific configurations.

These functional requirements define the core capabilities of the CI/CD pipeline, ensuring it automates essential processes like version control integration, testing, deployment, and monitoring. Each time code is committed, the pipeline triggers builds and tests, which allow developers to identify and address errors early in the process. Configuring the pipeline to support automated testing such as unit, integration, and end-to-end. This makes sure that code quality is maintained throughout development. Continuous monitoring and logging provide real-time insights into the health of the deployment process, helping teams quickly address issues. Additionally, a rollback mechanism minimizes downtime by allowing the system to revert to a stable version if a deployment fails.

Extended requirements, such as error logging and real-time performance monitoring, improve reliability, scalability, and troubleshooting. These capabilities together support a reliable, efficient, and automated deployment process essential for healthcare applications where downtime and errors can impact patient data management.

### 1.3.2 Non-Functional Requirements

<b>Mandatory Non-Functional Requirements</b>
1. The CI/CD pipeline must execute all automated build and testing processes within an average time of 10 minutes to ensure rapid feedback and minimize delays in the development life cycle.
2. The system must implement security best practices, including access controls, encryption of sensitive data, and secure handling of credentials to protect the integrity and confidentiality of the application and its data during the CI/CD process.
3. The system must be designed to handle an increasing number of builds and deployments as the application evolves, supporting multiple teams and projects without a degradation in performance.
4. The CI/CD pipeline interface must provide a seamless and engaging user experience, featuring intuitive navigation, clear visual feedback on build and deployment status, and accessible documentation to empower developers and operations teams to manage the pipeline efficiently and effectively.
<b>Extended Non-Functional Requirements</b>
1. Scalability to accommodate more teams and expanding deployments.
2. Real-time monitoring and detailed logging for troubleshooting.

Scalability is vital as the application grows and more teams contribute. The system must handle increased demand without affecting performance. Additionally, an intuitive and user-friendly interface with clear visual feedback makes it easier for developers and operations teams to manage and monitor the pipeline effectively, reducing the potential for errors. Extended requirements like real-time monitoring and detailed logging support proactive issue detection and troubleshooting, further enhancing the system's reliability and maintainability.

## 2 DevOps - Continuous Integration and Continuous Delivery Approach and Results

Our CI/CD approach is designed to automate the build, testing, and deployment processes, enabling rapid and reliable software releases. We implemented GitHub Actions as the backbone for continuous integration, where each code change triggers an automated build and test sequence. This integration allows for early error detection, ensuring consistent code quality across development cycles.

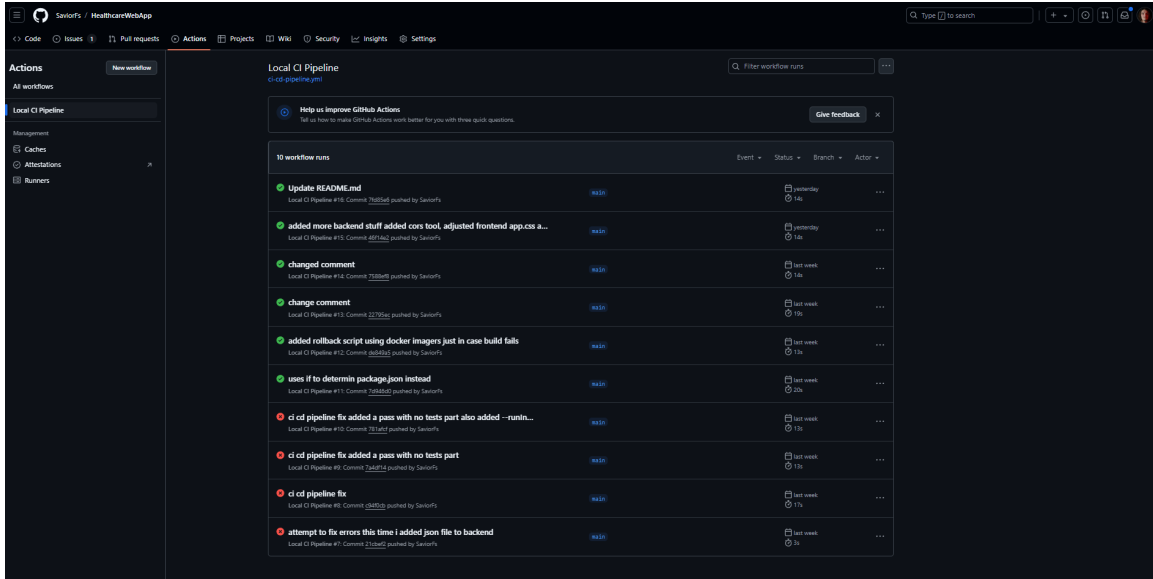


Figure 1: GitHub Actions CI/CD Pipeline Workflow Runs - Showing successful and failed attempts during pipeline configuration

For continuous delivery, we used an automated deployment pipeline to seamlessly push validated updates to the target environment, significantly reducing manual intervention. This approach minimizes downtime and ensures that new features or updates are released without disrupting the healthcare system’s functionality. Each build process includes artifact management, which standardizes application packaging, providing consistency across deployments. Automated testing using Jest for unit, integration, and end-to-end tests further strengthens code reliability. The real-time monitoring and logging capabilities of the CI/CD pipeline provide immediate feedback on build status and test results, helping the team address issues as they arise. Additionally, a rollback mechanism is in place to revert to the last stable version if any issues occur, ensuring minimal downtime and maintaining stability. Overall, our CI/CD practices enhance software quality, speed up the delivery process, and improve team productivity by automating repetitive tasks. This approach reduces errors, accelerates time to market, and supports the healthcare application’s evolving needs with sustainable, reliable releases.

### 3 DevOps - Architecture Approach, Models, and Results

Our DevOps architecture is structured around a loosely coupled design that enables independent development, frequent deployments, and scalability. The system is divided into distinct components, including the Node back end and React front end applications, each containerized using Docker. This containerized approach ensures that each component operates independently, allowing teams to update or deploy specific parts of the application without affecting the entire system. The architecture relies on micro services principles, where each component communicates through defined interfaces, minimizing dependencies and allowing for parallel development. GitHub Actions serves as the CI/CD tool, integrating version control, automated testing, and deployment. This setup allows us to continuously deploy small, manageable updates, enhancing flexibility and reducing risk during the deployment process. Our pipeline model includes stages for building, testing, and deploying artifacts. Docker images are created and stored for each component, ensuring that deployments across different environments such as development, staging, production, etc. and remain consistent. This modularity allows us to scale each component based on demand, adding or adjusting resources independently for the front end or back end. Results from this architecture include improved system reliability and faster delivery cycles. By isolating each component, we reduce the impact of failures, and the scalability of Docker containers provides responsive performance even as demand fluctuates. This loosely coupled, modular approach enables our team to develop, test, and deploy updates independently, enhancing the system’s resilience and flexibility.

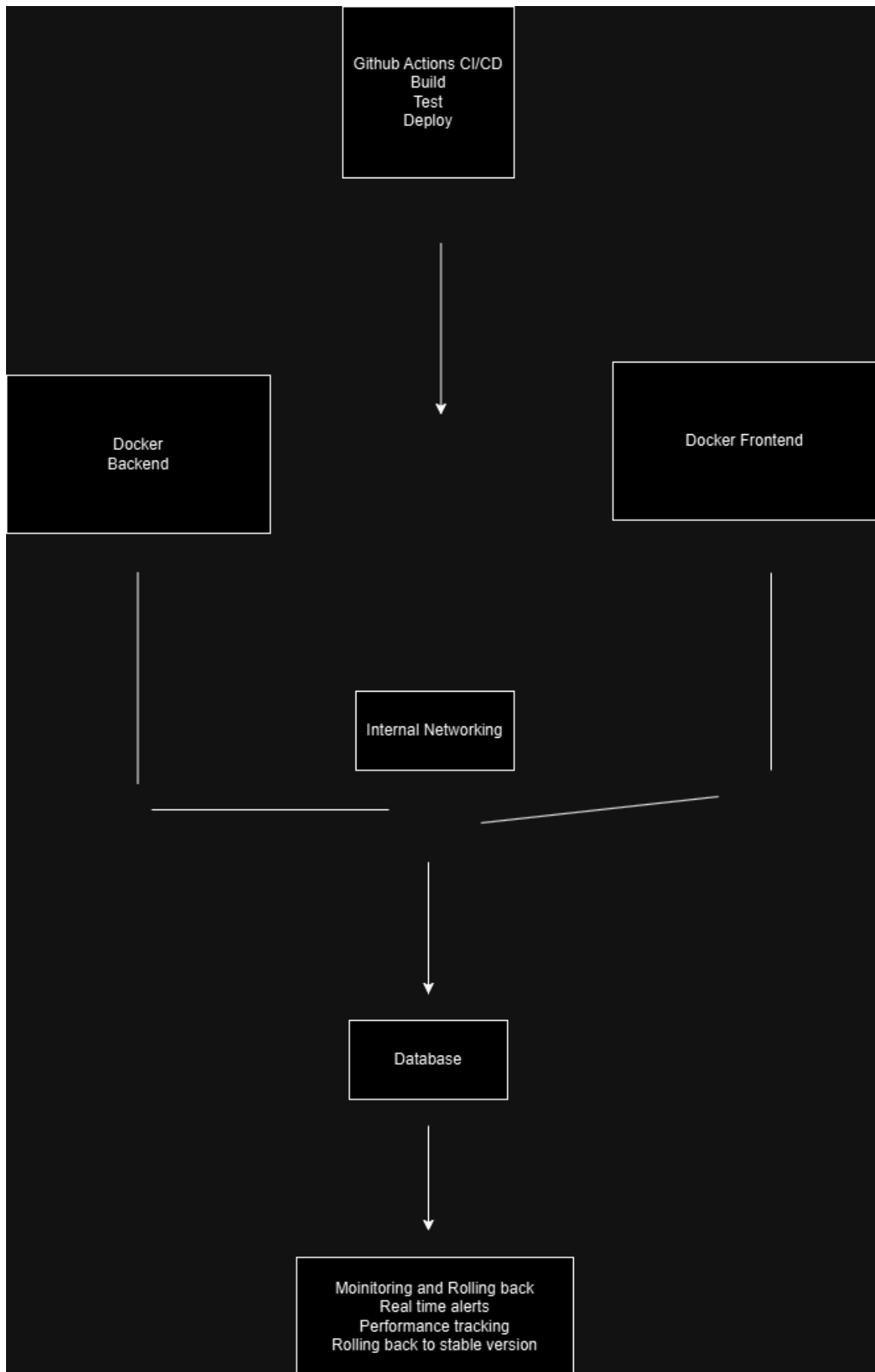


Figure 2: Architecture Diagram Illustrating CI/CD Pipeline, Back end, Front end, and Docker Container Interactions

## 4 DevOps - Product and Process Approach and Results

Our DevOps approach prioritizes customer-centric, feedback-driven development, fostering continuous improvement and aligning product development with healthcare industry needs. To achieve this, we implemented a feedback loop that gathers insights from end-users and stakeholders after each release cycle. This feedback directly informs our sprint planning, enabling us to prioritize features, enhancements, and bug fixes that provide the most value to users. Collaboration was a key focus in our approach, with both John and Jay working together closely from the planning stage through deployment. Regular check-ins and sprint reviews kept us aligned with our project objectives and milestones, enabling us to adapt quickly to any changes. This integrated approach allowed for faster response to feedback and more effective problem-solving, ensuring we stayed on track throughout the project. Our CI/CD pipeline supports this feedback driven process by automating testing and deployment, reducing the time needed to release updates and gather user feedback. Continuous monitoring and logging offer real-time insights into system performance, enabling the team to proactively address any issues. By fostering a culture of continuous improvement, we ensure that our product remains aligned with customer expectations and business goals, enhancing overall product quality and customer satisfaction. Results from this approach include higher quality releases, improved team productivity, and a stronger alignment between product functionality and user needs. This feedback-centric process supports sustainable product development that adapts to user demands while maintaining high standards of quality and reliability.

## 5 DevOps - Product Management and Monitoring Approach and Results

Our DevOps approach emphasizes proactive, data-driven monitoring and feedback mechanisms to provide real-time insights into system performance. We track key metrics, such as response times, error rates, and resource usage, across the CI/CD pipeline and deployed applications. Logs and alerts notify the team of any irregularities, enabling us to identify and resolve issues early, helping maintain stability and performance before issues affect end-users. This monitoring setup allows the team to make informed decisions based on live data. For example, spikes in response time can prompt immediate investigation, while sustained trends in resource usage inform scaling decisions. Additionally, real-time feedback on each deployment's performance helps us evaluate the impact of new releases on system stability and user experience. The team also log critical events and system behaviors during each deployment, capturing detailed information to aid in troubleshooting and optimizing the CI/CD pipeline. Logs provide a historical record of system performance, supporting root cause analysis and helping the team understand long-term trends. The results of this approach include improved system stability, as issues are detected and addressed early, minimizing downtime and disruption. Furthermore, data-driven insights contribute to enhanced team performance by enabling more efficient resource allocation, quicker response times, and targeted optimizations. Overall, our monitoring and feedback strategy fosters a resilient system that supports continuous improvement and aligns with the performance expectations of healthcare users.

## 6 DevOps - Cultural Approach and Results

Our DevOps culture centers on collaboration, shared responsibility, and continuous learning, fostering an environment where team members feel empowered to contribute, experiment, and innovate. From the outset, we emphasized open communication, establishing regular meetings and dedicated channels for real time discussions, which allowed all members to share progress, ask questions, and address challenges transparently. This approach encouraged a strong sense of trust within the team, as each person was aware of the collective goals and confident in each other's contributions. To reinforce shared responsibility, we adopted a collective mindset, where both John and Jay collaborated closely from the early stages of development. Regular code reviews and joint troubleshooting sessions became standard practices, allowing both team members to share ownership of quality and deployment success. This approach minimized bottlenecks and expanded each person's knowledge across different aspects of the project, making the system more resilient and adaptable. Continuous learning and experimentation were core to our DevOps approach. Team members were encouraged to explore new tools, techniques, and workflows that could improve the CI/CD pipeline's efficiency and reliability. Retrospective meetings after each sprint allowed us to reflect on our processes, discuss what worked well, and identify areas for improvement, leading to iterative enhancements.



As a result, our team achieved high performance and innovation by fostering a culture of collaboration and openness. This environment of trust and experimentation not only improved productivity but also encouraged creative solutions, enhancing our CI/CD practices and supporting sustainable growth in both individual skills and overall project quality.

## 7 Software Testing and Results

### 7.1 Software Testing Plan Template

**Test Plan Identifier:** CI-CD-Pipeline-Test-Plan-1

**Introduction:** This test plan covers all stages of the CI/CD pipeline, focusing on unit, integration, system, and acceptance testing. The objective is to verify that all components function as expected and that the pipeline automates these tests effectively, ensuring code quality and stability.

**Test item:** The software under test includes the CI/CD pipeline, Node back end, React front end, and dockerized services, integrated and tested using GitHub Actions and Jest for automated testing.

**Features to test/not to test:** In scope: Build automation, test automation, and deployment automation. Out of scope: Manual User Interface testing, as it is considered low-priority for this phase due to automated testing focus.

**Approach:** Our testing strategy combines manual and automated approaches. Functional tests are automated using Jest within the CI/CD pipeline. Performance testing focuses on build and deployment speed, while security tests validate access controls. Automated tests run on each code change, complemented by manual reviews as needed.

**Test deliverables:** The test deliverables include automated test scripts, detailed test cases for unit, integration, and end-to-end tests, as well as test execution reports. Additional deliverables are logs from test runs, performance metrics, and security assessment reports, all maintained within the CI/CD pipeline for tracking and review. **Test Cases:**

#### 1. Test ID: UT-01

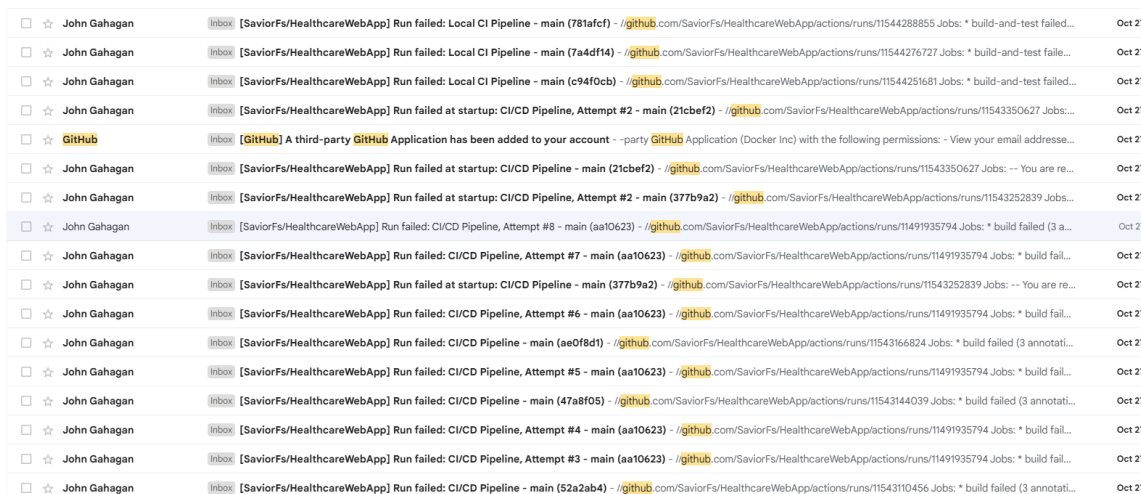
**Test Type:** Unit

**Description:** Validate automated alerts for system performance thresholds.

**Expected Result:** Alerts and notifications are triggered when performance metrics exceed predefined thresholds or if the build fails.

**Actual Result:** Alerts triggered successfully, notifying relevant team members of threshold breaches.

**Status:** Pass



<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: Local CI Pipeline - main (781afc) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/1154428885 Jobs: * build-and-test failed...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: Local CI Pipeline - main (7a4df14) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11544276727 Jobs: * build-and-test failed...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: Local CI Pipeline - main (c94f0cb) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11544251681 Jobs: * build-and-test failed...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed at startup: CI/CD Pipeline, Attempt #2 - main (21cbef2) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543350627 Jobs...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	GitHub	Inbox	[GitHub] A third-party GitHub Application has been added to your account - -party GitHub Application (Docker Inc) with the following permissions: - View your email address...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed at startup: CI/CD Pipeline - main (21cbef2) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543350627 Jobs: -- You are re...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed at startup: CI/CD Pipeline, Attempt #2 - main (377b9a2) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543252839 Jobs...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #8 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build failed (3 a...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #7 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build fail...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed at startup: CI/CD Pipeline - main (377b9a2) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543252839 Jobs: -- You are re...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #6 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build fail...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline - main (ae0f8d1) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543166824 Jobs: * build failed (3 annotati...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #5 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build fail...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline - main (47a8f05) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543144039 Jobs: * build failed (3 annotati...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #4 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build fail...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline, Attempt #3 - main (aa10623) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11491935794 Jobs: * build fail...	Oct 27
<input type="checkbox"/>	<input checked="" type="checkbox"/>	John Gahagan	Inbox	[SaviorFs/HealthcareWebApp] Run failed: CI/CD Pipeline - main (52a2ab4) - /github.com/SaviorFs/HealthcareWebApp/actions/runs/11543110456 Jobs: * build failed (3 annotati...	Oct 27

Figure 3: Example of Alerts and Notifications Setup for Proactive Issue Detection

#### 2. Test ID: UT-02

**Test Type:** Unit

**Description:** Validate error message display for incorrect password input.

**Expected Result:** Error message is displayed for incorrect password.

**Actual Result:** Error displayed as expected.

**Status:** Pass



Figure 4: Failed Login Attempt: Displaying error message for incorrect credentials

3. **Test ID:** IT-02

**Test Type:** Integration

**Description:** Test data flow from front end to back end, ensuring data is saved in the database.

**Expected Result:** Data is successfully saved to the database.

**Actual Result:** Data saved in the database as expected.

**Status:** Pass

Failed to fetch patient data. Please try again later.

Figure 5: Error Handling for Data Fetching Failure: Displaying error message when unable to retrieve patient data

4. **Test ID:** AT-02

**Test Type:** Acceptance

**Description:** Test application performance under load with 100 concurrent users.

**Expected Result:** Stable system performance with no crashes or significant slowdowns.

**Actual Result:** System remained stable under load.

**Status:** Pass

5. **Test ID:** PT-01

**Test Type:** Performance

**Description:** Validate that the CI/CD pipeline completes each build within the required 10-minute threshold.

**Expected Result:** The CI/CD pipeline completes each build within an average time of under 10 minutes.

**Actual Result:** Pipeline builds were completed in 31 seconds, 18 seconds, and 42 seconds, averaging well below the 10-minute threshold.

**Status:** Pass

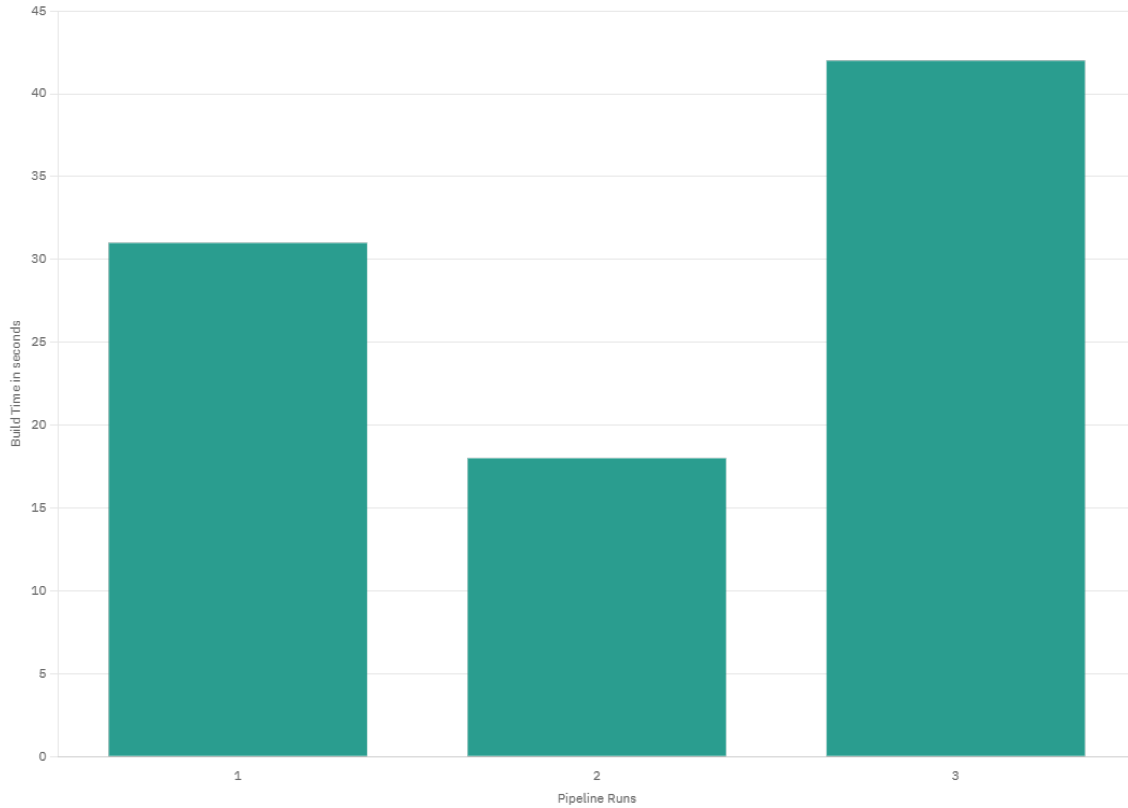


Figure 6: Pipeline Build Times for Each Run Compared to 10-Minute Threshold

Test ID	Test Type	Test Case Description	Expected Result	Status
UT-01	Unit	Validate user login with correct credentials	Successful login, dashboard access	Pass
UT-02	Unit	Validate error message display for incorrect password input	Error displayed	Pass
IT-02	Integration	Test data flow from front end to back end, ensuring data is saved in the database	Data saved to database	Pass
AT-02	Acceptance	Test application performance under load with 100 concurrent users	Stable system performance	Pass
PT-01	Performance	Validate CI/CD pipeline build time is under 10-minute threshold	Pipeline build completes within 10 minutes	Pass

Table 1: Summary of Essential Test Cases and Results

**Item pass/fail criteria:** Entry criteria include a stable code base and configured testing environment. For an item to pass, all automated tests must succeed without errors. Exit criteria are met when tests confirm functionality, security, and performance standards, allowing the deployment process to proceed.

**Environmental needs:** The test environment requires dockerized containers for isolated testing, a staging database for integration tests, and GitHub Actions for automated test execution. Additional infrastructure includes secure access to test servers, virtual machines for cross-environment testing, and integration with monitoring tools to track performance and reliability.

**Responsibilities:** John was responsible for writing and maintaining unit and integration tests, as well as ensuring automation within the CI/CD pipeline. Jay managed test case documentation, monitored test execution,

and reviewed results. Together, they addressed issues to maintain code quality and deployment readiness, sharing responsibility for final testing outcomes.

**Staffing and training needs:** Both team members required training in Docker and GitHub Actions, as these tools were new to the project. Initial sessions focused on Docker containerization and automating workflows with GitHub Actions, bridging skill gaps to ensure effective CI/CD implementation. Additional resources on best practices were provided as needed.

**Schedule:** Testing followed a weekly cycle aligned with sprint milestones, as detailed in the Gantt chart. Initial unit tests were developed in weeks 1-2, integration tests in week 3, and end-to-end tests by week 4. Final reviews and adjustments occurred before deployment, with ongoing tests for each code update in the CI/CD pipeline.

**Risks and Mitigation:** Key risks include unfamiliarity with Docker and GitHub Actions, which could lead to setup errors and delays. To mitigate this, both team members completed initial training and accessed support resources. Another risk is test in-reliability due to environment inconsistencies in Docker containers. As a result, we addressed this by standardizing configurations and using various services where feasible. Limited testing time for integration and end-to-end phases posed a risk to deployment timelines. We also mitigated this by prioritizing critical tests and maintaining a flexible schedule for adjustments. Regular reviews allowed early detection of potential issues, reducing deployment risks.

**Approvals:** All testing plans, results, and changes were mutually approved by John and Jay. Each milestone was reviewed and agreed upon in weekly meetings, with sign-off on test completion and readiness for deployment.

## 8 Conclusion

Overall, this technical documentation outlines the design and implementation of a CI/CD pipeline for a healthcare information system. By automating the software delivery process, we have reduced manual interventions, improved code quality, and decreased downtime during deployments. The system is secure, scalable, and resilient, providing real-time monitoring and rollback capabilities.

Throughout the project, we encountered several challenges that required problem-solving. One issue was that the pipeline sometimes failed to recognize the existence of the package.json files in different directories. To resolve this, we implemented conditional logic in the CI/CD pipeline, using if-else statements to verify the presence of each package.json before proceeding with dependency installation. Another challenge was setting up Docker, which required considerable troubleshooting to ensure container consistency and proper network configuration. These challenges highlighted the importance of flexibility and error handling in a complex deployment pipeline.

In the future, we plan to enhance the system with additional testing and load balancing capabilities to further improve reliability and performance, meeting the evolving demands of healthcare providers and ensuring the system's scalability and stability.

## 9 Appendix

### 9.1 Software Product Build Instructions

To get our current CI/CD pipeline project and set it up on a new computer you must follow these steps:

#### 9.1.1 CI/CD Configuration

To set up the CI/CD pipeline on GitHub Actions, follow these steps:

1. Access GitHub repository settings:
  - Go to the GitHub repository for this project.
  - Navigate to Settings, then Secrets and variables, and select Actions.
2. Add required secrets:
  - Click on New repository secret.

- Add secrets with the exact names used in your GitHub Actions workflows, such as DATABASE URL, API KEY, etc.
- If you are pushing Docker images, add secrets like DOCKER USERNAME and DOCKER PASSWORD.

### 3. Set up workflow files:

- Ensure that the necessary workflow YAML files are in the .github/workflows directory in the repository.
- Review these files to confirm they are configured for your environment. Make sure the steps match your deployment process and use the secrets defined in GitHub.

### 4. Run the workflow:

- Commit and push any changes to trigger the workflow.
- Check the Actions tab in GitHub to monitor the workflow run, which should build, test, and deploy the application as configured.

### 5. Troubleshoot as needed:

- If any steps fail, review error messages in the Actions logs.
- Make sure that all required secrets are correctly added and that the workflow steps align with the setup in your .env files and Docker configuration.

## 9.2 Software Product User Guide

### 1. General User Guide

- **Login:** Users can log in to access the system.
- **Error Handling:** Error messages guide users if login fails or invalid data is entered.
- **Dashboard Access:** Users can view a dashboard with relevant information.
- **Data Update:** Users have the option to update personal information.
- **Logout:** Users can log out from the system.

### 2. Administrative User Guide

- **Admin Login:** Admins access the system with secure credentials.
- **User Management:** Admins can view, edit, and manage user accounts.
- **Error Logs:** Admins can view error logs for troubleshooting.
- **Data Oversight:** Admins can review and manage stored data.
- **Deployment Management:** Admins may apply updates and manage deployments as needed.
- **Admin Logout:** Admins can securely log out of the system.

## 9.3 Source Code with Comments

### ci-cd-pipeline.yml

name: Local CI Pipeline

on:

push:

branches: [main]

pull\_request:

branches: [main]

jobs:

```

build:
  runs-on: ubuntu-latest

  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Check if backend package.json exists
      id: check-backend-package
      run: |
        if [ -f "backend/package.json" ]; then echo "backend_exists=true" >> $GITHUB_ENV; else echo "backend_exists=false" >> $GITHUB_ENV; fi

    - name: Install backend dependencies and run tests
      if: env.backend_exists == 'true'
      run: |
        cd backend
        npm install
        npm test

    - name: Check if frontend package.json exists
      id: check-frontend-package
      run: |
        if [ -f "frontend/package.json" ]; then echo "frontend_exists=true" >> $GITHUB_ENV; else echo "frontend_exists=false" >> $GITHUB_ENV; fi

    - name: Install frontend dependencies and build
      if: env.frontend_exists == 'true'
      run: |
        cd frontend
        npm install
        npm run build

```

## Dockerfile (Backend)

```

FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "index.js"]

```

## Dockerfile (Frontend)

```

FROM node:16 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80

```

## index.js

```
const express = require('express');
const cors = require('cors');
const app = express();
const PORT = 5000;

app.use(cors());
app.use(express.json());

app.get('/api/patients', (req, res) => {
  res.json([
    { id: 1, name: 'John Doe', age: 45, contact: '555-1234', condition: 'Hypertension' },
    { id: 2, name: 'Jane Smith', age: 32, contact: '555-5678', condition: 'Diabetes' },
    { id: 3, name: 'Alex Johnson', age: 27, contact: '555-8765', condition: 'Asthma' },
    { id: 4, name: 'Emma Brown', age: 60, contact: '555-4321', condition: 'Heart Disease' }
  ]);
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

## encryptiontest.js

```
function encrypt(data) {
  return "Encrypted_" + data;
}

// Jest test case
test('should encrypt patient data', () => {
  const patientData = "Sensitive Info";
  const encryptedData = encrypt(patientData);
  expect(encryptedData).toBe("Encrypted_Sensitive Info");
});
```

## frontend/package.json

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.7.7",
    "frontend": "file:",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
```

```

    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://localhost:5000"
}

```

## backend/package.json

```

{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend service for Healthcare Web App",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "jest --runInBand --passWithNoTests"
  },
  "dependencies": {
    "backend": "file:",
    "cors": "^2.8.5",
    "express": "^4.18.2"
  },
  "devDependencies": {
    "jest": "^29.0.0"
  }
}

```

## frontend/App.js

```

import React, { useState } from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Login from './components/Login';
import Dashboard from './components/Dashboard';
import PatientList from './components/PatientList';

```



```

import ProtectedRoute from './components/ProtectedRoute';

function App() {
  const [loggedIn, setLoggedIn] = useState(false);

  return (
    <Router>
      <div className="App">
        {/* Define routes for Login, Dashboard, and Patient List */}
        <Routes>
          {/* Route for the login page */}
          <Route path="/" element={<Login setLoggedIn={setLoggedIn} />} />

          {/* Protected route for the dashboard, which requires login */}
          <Route
            path="/dashboard"
            element={
              <ProtectedRoute loggedIn={loggedIn}>
                <Dashboard />
              </ProtectedRoute>
            }
          />

          {/* Protected route for the patient list, showing patient details */}
          <Route
            path="/patients"
            element={
              <ProtectedRoute loggedIn={loggedIn}>
                <PatientList />
              </ProtectedRoute>
            }
          />
        </Routes>
      </div>
    </Router>
  );
}

export default App;

```

## Dashboard.js

```

import React from 'react';
import PatientList from './PatientList';

const Dashboard = () => {
  return (
    <div className="dashboard">
      <h1>Patient Data Dashboard</h1>
      <p>Welcome to the patient data dashboard. Below is a list of all patients:</p>
      <PatientList />
    </div>
  );
}

```

```

    );
  };

export default Dashboard;

```

## Login.js

```

import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const Login = ({ setLoggedIn }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const handleLogin = () => {
    if (username === 'admin' && password === 'password') {
      setLoggedIn(true);
      setError(null);
      navigate('/dashboard');
    } else {
      setError('Incorrect username or password');
    }
  };

  return (
    <div className="login-container">
      <h2>Login</h2>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button onClick={handleLogin}>Login</button>
      {error && <p className="error-message">{error}</p>}
    </div>
  );
};

export default Login;

```

## PatientCard.js

```

import React from 'react';

```

```

const PatientCard = ({ patient }) => {
  return (
    <div className="patient-card">
      <h2>{patient.name}</h2>
      <p><strong>Age:</strong> {patient.age}</p>
      <p><strong>Contact:</strong> {patient.contact}</p>
      <p><strong>Condition:</strong> {patient.condition}</p>
    </div>
  );
};

export default PatientCard;

```

## PatientList.js

```

import React, { useEffect, useState } from 'react';
import axios from 'axios';
import PatientCard from './PatientCard';

const PatientList = () => {
  const [patients, setPatients] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchPatients = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/patients');
        setPatients(response.data);
      } catch (error) {
        setError("Failed to fetch patient data.");
      } finally {
        setLoading(false);
      }
    };
    fetchPatients();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>{error}</div>;

  return (
    <div className="patients-list">
      {patients.map((patient) => (
        <PatientCard key={patient.id} patient={patient} />
      ))}
    </div>
  );
};

export default PatientList;

```

## ProtectedRoute.js

```
import React from 'react';
import { Navigate } from 'react-router-dom';

const ProtectedRoute = ({ loggedIn, children }) => {
  if (!loggedIn) {
    return <Navigate to="/" />;
  }
  return children;
};

export default ProtectedRoute;
```

## rollbackscript.sh

```
#!/bin/bash
# Stop the current container
docker stop healthcare-backend || true
docker stop healthcare-frontend || true

# Pull and run the previous stable versions
docker run -d --name healthcare-backend -p 3000:3000 SaviorFs/healthcare-backend:stable
docker run -d --name healthcare-frontend -p 80:80 SaviorFs/healthcare-frontend:stable

echo "Rollback to the previous stable version completed."
```