IT UNIVERSITY OF COPENHAGEN

DevOps, Software Evolution and Software Maintenance (Spring 2023)

Course Code: BSDSESM1KU

May 2, 2023

Group members

Gustav Valdemar Metnik-Beck gume@itu.dk
Nikolaj Worsøe Larsen niwl@itu.dk
Simon Johann Skødt sijs@itu.dk
Victor Møller Wauvert Brorson vibr@itu.dk

Number of characters *including* spaces: 2805
Number of characters *excluding* spaces: 2409

Contents

1 Introduction

Throughout the course *DevOps, Software Evolution and Software Maintenance*, the legacy Python-based Twitter application was migrated to a modern .NET-based application. Following the initial migration, the task at hand was to handle simulated users using DevOps principles and practices in day-to-day development [1]. The system was designed to support fundamental functionalities of a Twitter application, such as user registration, login, post creation, follow requests, and viewing others' posts. The system was operated on a weekly basis, with new methods and strategies integrated into the system, such as continuous integration, delivery and deployment, logging, monitoring, and scalability, thereby enabling a comprehensive understanding of the importance of DevOps in managing software systems.

2 System Perspective

2.1 System Design

2.2 System Architecture

2.2.1 Subsystems

2.3 System Dependencies

2.4 Current State

2.5 Licence Compatability

3 Developing Process

3.1 Developer Interaction

3.2 Team Organisation

3.3 CI/CD

To facilitate the continues integration of new code by all developers, several workflows/pipelines have been implemented to ensure that each pull request (PR) is subjected to various checks, including building, testing, and code quality analysis to mitigate the risk of the service to break. This could happen due to defects, vulnerabilities, and other undesirable coding practices. Once these checks are successfully completed, the PR is deemed safe for merging into the master branch, which initiates a deployment pipeline.

3.3.1 build-test.yml

This workflow performs the build and testing of the entire stack of the service, including the backend and fronted. The pipeline is triggered on push event to any branch and includes two jobs:

build-test-backend, which runs on an ubuntu 20.04 machine and includes the following steps:

1. Checkout the repository
2. Set up .NET version 7.0.x
3. Restore dependencies
4. Build the MiniTwit backend
5. Run backend tests

build-test-frontend, which runs on an ubunto 20.04 machine and includes the following steps:

1. Checkout the repository
2. Install Node.js version 18 and dependencies for caching

3. Install dependencies for the MiniTwit frontend
4. Build the MiniTwit frontend
5. Run frontend tests using Playwright, in a docker container started by docker-compose -f docker-compose.ui.yml up -d
6. Stop the UI service by shutting down the Docker containers created with docker-compose

3.3.2 codeql.yml

Pipeline for codeQL (C sharp)

3.3.3 eslint.yml

Pipeline for checking TS code

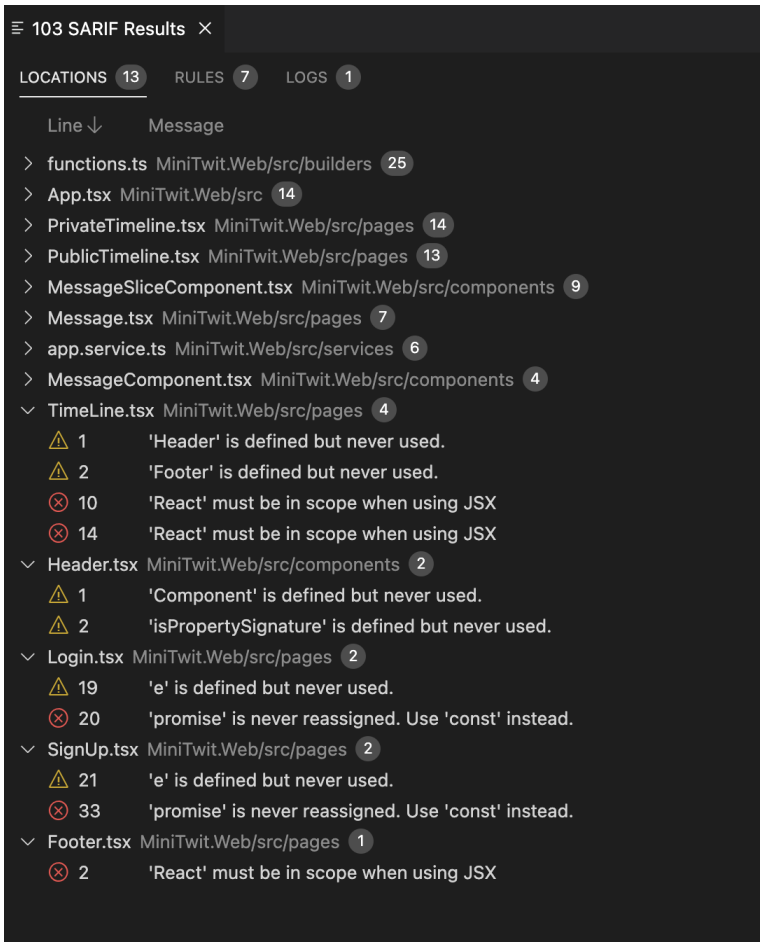

Figure 3.1: Eslint scan - Report

3.3.4 snyk-security.yml

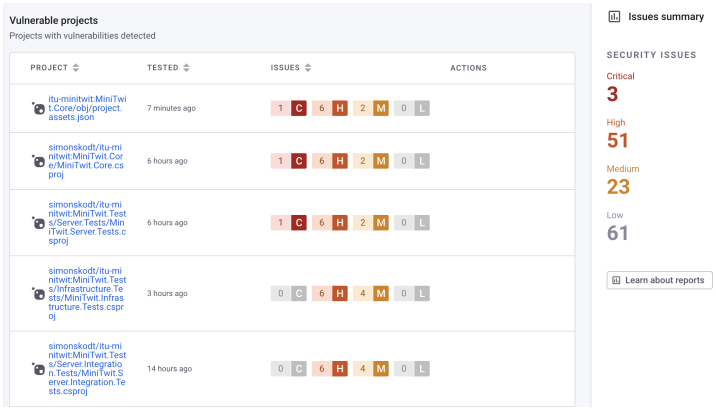

Figure 3.2: Snyk Security - Report

3.3.5 SonarCloud & CodeClimate

Integration with sonarcloud and codeclimate

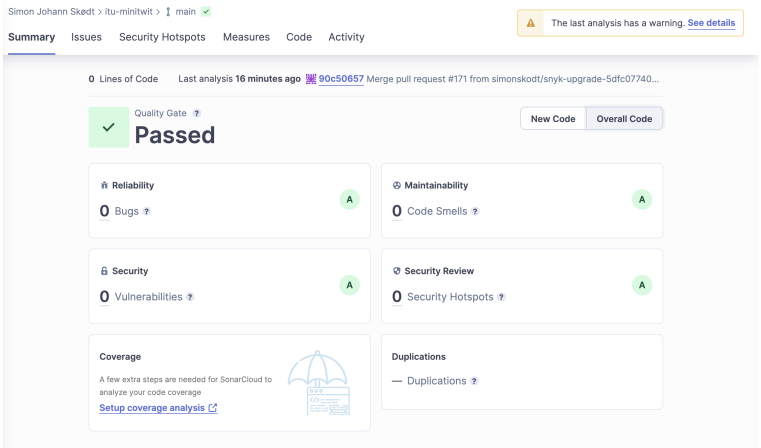

Figure 3.3: Sonar Cloud - Report

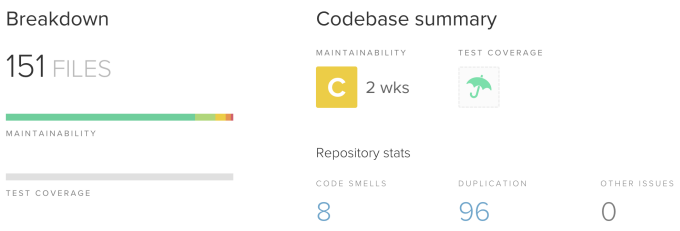

Figure 3.4: Code Climate - Report

3.3.6 continuous-deployment.yml

Deploytment pipeline (only on push)

3.4 Repository Organisation

3.5 Branching Strategy

3.6 Monotoring

3.7 Logging

3.8 Security

3.9 Scaling and Load Balancing

4 Lessions

4.1 Evolution and Refactoring

4.2 Maintenance

4.3 Operation

4.4 DevOps Style

5 Conclusion

References

[1] Renée McCauley et al. “Debugging: a review of the literature from an educational perspective”. In: *Computer Science Education* (2008), p. 67.

Appendices

A Test