

REPUBLIC OF CAMEROON

Peace-Work-Fatherland

MINISTER OF HIGHER
EDUCATION

FACULTY OF ENGINEERING
AND TECHNOLOGY



REPUBLIQUE DU CAMEROON

PAIX-Travail-Patrie

MINISTRE DE L'ENSEIGNEMENT
SUPERIEUR

FACULTE DE L'INGENIERIE
ET TECHNOLOGIE

***** UNIVERSITY OF BUEA *****



DEPARTMENT OF COMPUTER ENGINEERING

COURSE TITLE/CODE: SOFTWARE QUALITY TOOLS – CEF431

COURSE INSTRUCTORS: Dr. HUGUES MARIE KAMDJOU & Dr. SOP DEFFO Lionel

PROJECT TITLE:

CONTINUOUS INTEGRATION PIPELINE WITH JENKINS

Presented by:

S/N	NAMES	MATRICULE
1	ARREY ABUNAW REGINA EBAI	FE22A152
2	ARREY-TABOT PASCALINE	FE22A151
3	ATANKEU TCHAKOUTE ANGE NATANAHEL	FE22A158
4	ATIKU BORIS-BEN MFAI	FE22A159
5	AV'NYEHBEUHKOH EZEKIEL HOSANA	FE22A160
6	AWA ERIC ANGELO JUNIOR	FE22A162
7	AYUK NESTOR EYONG	FE22A164
8	BEH CHU NELSON	FE22A170
9	BEKONO AKONO MARTHE LUNYA LAWERICA	FE22A171
10	BELLAH LOVETTE MANYI	FE22A172
11	KONINGS AKOKO	FE24A623

Table Of Content

Problem Statement.....	4
Objectives	4
Main Objective	4
Specific Objectives.....	4
Methodology	4
Steps for Creating the CI Testing Pipeline	5
1. Environment Setup	5
o Visual Studio Code	5
o Pytest for testing	5
o Flake8 for code quality checks	5
o Jenkins for CI.....	5
o GitHub for version control and collaboration.....	5
o Ngrok for secure tunneling	5
2. Repository Management.....	5
3. Version Control and Collaboration.....	5
4. CI Pipeline Configuration in Jenkins	5
4.2 Pipeline Configuration Steps:	5
1. Basic Configuration:.....	5
2. Build Triggers:	6
3. Advanced Pipeline Configurations:	7
4. Jenkins file Creation:	9
5. Execution of Tests	11
6. Monitoring and Results Analysis	11
Test Plan	12
Participants.....	12
Objectives	12
Scope of Testing	13
Resource Planning.....	13
Test Strategy	13
Build Triggers	14
Advanced Pipeline Configurations.....	14
Schedule.....	15
Activity: Creation of a remote repository on GitHub	15
Activity: Creation of a CRUD-APP	15
Activity: Independent Creation of unit test cases	15
Activity: Creation of the Jenkins pipeline	15
Activity: Documentation	15
Activity: Project Closure.....	15
Risk Management	15
Potential Threats.....	15
Mitigation Strategies.....	16
Pipeline Metrics	16
Test Procedure	17
Test Setup	17

Unit Test	17
Execution Instructions.....	17
Unit Test	17
Integration Test	18
Requirements for Integration Test.....	18
Expected Results	18
Unit Test	18
Integration Test	20
Error Handling	20
Common Errors.....	20
Mitigation Strategies.....	20
Test Report.....	21
Test Cases	21
Results.....	21
Summary.....	21
IV. Discussion.....	22
V. Conclusion.....	22

I. Introduction

In modern software development, Continuous Integration (CI) is critical for automating the build and testing process to ensure code quality and reliability. This report focuses on setting up a CI pipeline using Jenkins to automate tests for a project hosted on GitHub. The configuration emphasizes regular and event-driven testing, comprehensive coverage analysis, and actionable recommendations to address gaps in testing.

II. Project Description

Problem Statement

Effective CI pipelines are essential for maintaining software quality, but improper configurations or inadequate test coverage can lead to undetected issues, delayed releases, and decreased efficiency.

Objectives

Main Objective

To set up a Continuous Integration (CI) pipeline using Jenkins that runs automated tests for a GitHub-hosted project.

Specific Objectives

1. Configure Jenkins to automate testing processes using periodic and event-driven triggers.
2. Ensure the pipeline operates on the main branch to validate new code before merging.
3. Analyze test coverage and enhance testing for under-tested areas of the project.
4. Document the test plan, test procedure, and test report for transparency and reproducibility.

III. Methodology for Creating a CI Testing Pipeline

Overview

The methodology outlined below details the steps taken to create a Continuous Integration (CI) testing pipeline for the CRUD-based web application. The goal was to ensure efficient testing and integration of the application, maintaining high software quality throughout the development process.

Steps for Creating the CI Testing Pipeline

1. Environment Setup

- **Technologies Used:**
 - Visual Studio Code
 - Python and its dependencies (as listed in requirements.txt)
 - Pytest for testing
 - Flake8 for code quality checks
 - Jenkins for CI
 - GitHub for version control and collaboration
 - GitHub link: <https://github.com/ericBlack1/Software-Quality-Tools-Project>
 - Ngrok for secure tunneling

2. Repository Management

- A GitHub repository named "Software-Quality-Tool-Project" was created to host the application code.
- Team members were added as collaborators, enabling them to contribute to the project.

3. Version Control and Collaboration

- Git was used for version control. Each team member created unit tests for at least one function in the application, ensuring they adhered to Flake8 formatting standards.
- Upon successful testing, changes were pushed to the remote repository, and pull requests were created for review by the main administrator.

4. CI Pipeline Configuration in Jenkins

4.1 Pipeline Creation

- A new pipeline was created in Jenkins by navigating to the dashboard, selecting "New Item," and choosing "Pipeline."

4.2 Pipeline Configuration Steps:

1. **Basic Configuration:**
 - Ensured no concurrent builds were allowed.
 - Set the project path to the GitHub repository.

Tableau de bord > Integration_test > Configuration

Description

Integration testing on a CRUD_APP

Texte brut [Prévisualisation](#)

☐ Ce build a des paramètres ?

☐ Run the build inside Docker containers

☒ Do not allow concurrent builds

☐ Abort previous builds ?

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url ?

<https://github.com/At-an/Software-Quality-Tools-Project.git/>

2. Build Triggers:

- Configured periodic builds to trigger daily at 6 AM.
- Configured GitHub webhooks for SCM.

4+8..

Build Triggers

☐ Build when a change is pushed to GitLab. GitLab webhook URL:
http://localhost:8000/project/Integration_test

☐ Construire après le build sur d'autres projets ?

☒ Construire périodiquement ?

Planning ?

0 6 * * *

⚠ Etaler la charge de façon régulière en utilisant 'H 6 * * *' plutôt que '0 6 * * *'
Aurait été lancé à dimanche 5 janvier 2025, 06:00:01 heure normale d'Afrique de l'Ouest; prochain à lundi 6 janvier 2025, 06:00:01 heure normale d'Afrique de l'Ouest.

☐ GitHub Branches

☐ GitHub Pull Request Builder

☐ GitHub Pull Requests ?

☒ GitHub hook trigger for GITScm polling ?

☐ Monitor Docker Hub/Registry for image changes ?

☐ Scrutation de l'outil de gestion de version ?

☒ Période d'attente ?

3. Advanced Pipeline Configurations:

- **SCM:** Set to Git.
- **Repository Path:** https://github.com/At-an/Software-Quality-Tools-Project.git.
- **Credentials:** Added necessary credentials for the pipeline.
- **Branch Specifier:** Set to /main, ensuring testing is performed on the main branch.
- **Script Path:** Defined the path to the Jenkinsfile containing pipeline commands.

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/At-arySoftware-Quality-Tools-Project.git

Credentials ?

Jenkins pipeline for integration testing

+ Ajouter

Avancé

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Navigateur de la base de code ?

(Auto)

Additional Behaviours

Ajouter

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

Pipeline Syntax

Sauvegarder Appliquer

4. Jenkins file Creation:

- A Jenkinsfile was created in the local project directory, outlining various stages of the pipeline, including:
 - Checkout SCM
 - Install dependencies
 - Code quality checks
 - Generate requirements.txt
 - Generate journal.html
 - Code coverage reports
 - Post-stage reporting for pipeline status

```
pipeline {
    agent any
    tools {
        git 'Git(Default)'
    }

    environment {
        PYTHON_VERSION = '3.12.5'
        PYTHON_EXE = 'C:\\Users\\ANGE\\AppData\\Local\\Programs\\Python\\Python312\\python.exe'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/main']],
                    extensions: [[$class: 'CleanCheckout']],
                    userRemoteConfigs: [[
                        credentialsId: 'ange',
                        url: 'https://github.com/At-an/Software-Quality-Tools-Project.git'
                    ]]
                ])
            }
        }

        stage('Install Dependencies') {
            steps {
                script {
                    bat """
                        ${PYTHON_EXE} -m pip install --user ^
                    """
                }
            }
        }
    }
}
```

```

stage('Code Quality Check') {
    steps {
        script {
            dir('C:\\ProgramData\\Jenkins\\.jenkins\\workspace\\Integration_test') {
                bat """
                    ${PYTHON_EXE} -m flake8 . --max-line-length=120
                """
            }
        }
    }
}

stage('Run Integration Tests') {
    steps {
        script {
            bat """
                REM Create directories for test results
                mkdir test-results
                mkdir coverage-reports

                REM Run tests with coverage
                ${PYTHON_EXE} -m pytest tests\\ ^
                    --verbose ^
                    --junitxml=test-results\\junit.xml ^
                    --cov=. ^
                    --cov-report=xml:coverage-reports\\coverage.xml ^
                    --cov-report=html:coverage-reports\\html
            """
        }
    }
}

stage('Generate Requirements') {
    steps {
        script {
            bat """
                ${PYTHON_EXE} -m pip freeze > requirements.txt
            """
        }
    }
}

```

```

post {
    always {
        bat 'powershell -Command "Get-ChildItem -Path . -Filter TEST-*.xml -Recurse | Format-Table"'
        junit 'test-results/*.xml'
        recordIssues enabledForFailure: true, tools: [flakes()]
        publishCoverage adapters: [coberturaAdapter('coverage-reports/coverage.xml')]
        publishHTML([
            allowMissing: false,
            alwaysLinkToLastBuild: true,
            keepAll: true,
            reportDir: 'coverage-reports/html',
            reportFiles: 'index.html',
            reportName: 'Coverage Report'
        ])
    }
    success {
        echo 'Integration tests completed successfully!'
    }
    failure {
        echo 'Integration tests failed!'
    }
    cleanup {
        cleanWs()
    }
}
}

```

5. Execution of Tests


- Unit tests were executed using Pytest, and code quality was checked using Flake8.
- Upon pushing changes to the repository, the configured pipeline was triggered automatically, allowing for continuous integration testing.

6. Monitoring and Results Analysis


- The status of the pipeline was monitored via Jenkins, providing insights into test results, execution time, and coverage reports.

Tableau de bord > Integration_test > #93


#93 (5
 janv.
 2025,
 09:52:14)

 Ajouter une description


Conserver cette construction sans limite de temps

 Lancé par une alarme périodique


Démarrée il y a 20 mn.
 A duré 1 mn 24 s


 This run spent:


- 0,1 s waiting;
- 1 mn 24 s build duration;
- 1 mn 24 s total from scheduled to completion.


 **Revision:** 5d6f55fa9a1687af277deb23c15d4babc067e772
Repository: <https://github.com/At-an/Software-Quality-Tools-Project.git>

- refs/remotes/origin/main

 [Résultats des tests](#) (aucune erreur)

 Flake8: No warnings ⓘ

- No issues for 56 builds, i.e. since build:  #38

 [Coverage Report](#)

- [Project Coverage:](#)

Test Plan

Participants

- AWA ERIC ANGELO
- ATANKEU TCHAKOUTE ANGE
- AV'NEUBEUH HOSANA
- ARREY REGINA EBAI
- ARREY PASCALINE
- AYUK NESTOR
- ATIKU BORIS
- BEH CHU NELSON
- BEKONO AKONO LAWRICA
- BELLA LOVETTE
- KONINGS AKOKO

Objectives

This testing process is aimed at testing and performing continuous integration testing on a CRUD-based web app to ensure proper and clean code development, increasing software quality, and ensuring continuous app development over time.

Scope of Testing

This test process will cover all the functions in the views, app, and auth modules of the application. Some limitations of the testing include that it will not test the functions in the models module of our application simply because it is a serverless application.

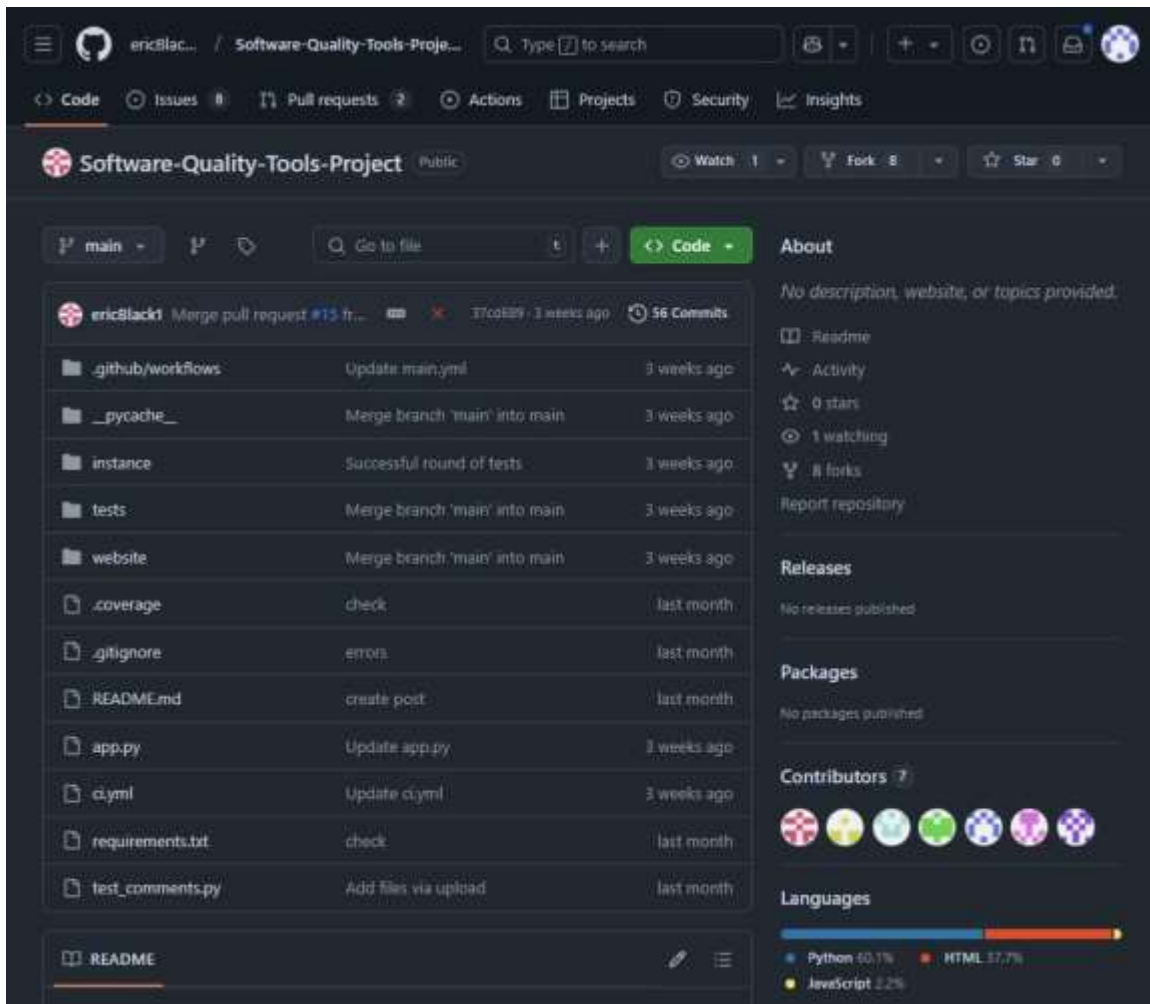
Resource Planning

We made use of several technologies (VSCode, Python and its dependencies mentioned in the requirements.txt file, pytest, flake8, Jenkins with a Jenkinsfile, GitHub and GitHub webhooks, ngrok for creating tunnels required to access a public network safely, and much more).

Each team member was responsible for writing a test case for at least one function in the application.

Test Strategy

We used Git and GitHub for version control and collaboration. After the app was developed and pushed to the GitHub repository (Software-Quality-Tool-Project), each member downloaded and ran it locally. The main administrator of the repository added other members as collaborators and assigned test issues to them. Each member wrote unit test cases for at least one function and ensured that it runs correctly and aligns with flake8 syntax for formatting code, using pytest as the testing framework. After successful completion of a test, a push is done to the remote repository, and a pull request is created where the main administrator checks it out and decides to merge the code if it is okay or reassign this issue to the same member.



After all unit test cases were written for the application, a CI testing pipeline was created and tested on Jenkins through the following steps:

1. Install Jenkins and access the service at localhost:8000.
2. Create an account and install all required dependencies for the pipeline and project in the Jenkins plugin.
3. Configure the pipeline (details provided in the original document).

Build Triggers

- Periodic build trigger at 6 AM daily.
- GitHub pull requests trigger.
- GitHub webhook for Git SCM.

Advanced Pipeline Configurations

- SCM: Git
- Repository: <https://github.com/At-an/Software-Quality-Tools-Project.git>.

- **Credentials:** Required credentials for the pipeline.
- **Branch Specifier:** /main.
- **Script path:** Jenkins.

Schedule

Activity: Creation of a remote repository on GitHub

- **Objective:** Create a GitHub Repository and establish collaborators.
- **Participants:** Awa Eric Angelo
- **Timeline:** 10/12/24 - 11/12/24

Activity: Creation of a CRUD-APP

- **Objective:** Create and implement the CRUD-APP using Python.
- **Participants:** Awa Eric Angelo.
- **Timeline:** 11/12/24 - 13/12/24

Activity: Independent Creation of unit test cases

- **Objective:** Create a test strategy and perform unit testing on each function.
- **Participants:** The Whole Team
- **Timeline:** 14/12/24 - 16/12/24.

Activity: Creation of the Jenkins pipeline

- **Objective:** Create a Jenkins pipeline to perform integration testing.
- **Participants:** Atankeu Tchakoute Ange and other team members.
- **Timeline:** 20/12/24 - 30/12/24.

Activity: Documentation

- **Objective:** Document the project.
- **Participants:** Atankeu Tchakoute Ange, Beh Chu Nelson, and Arrey Regina.
- **Timeline:** 02/01/25 - 05/01/25

Activity: Project Closure

- **Objective:** Close the project and appreciate the efforts of every team member.
- **Timeline:** 07/01/25

Risk Management

Potential Threats

1. Lack of communication and collaboration.
2. Insufficient knowledge about integration testing planning, design, and implementation.
3. Insufficient internet connection to complete tasks.

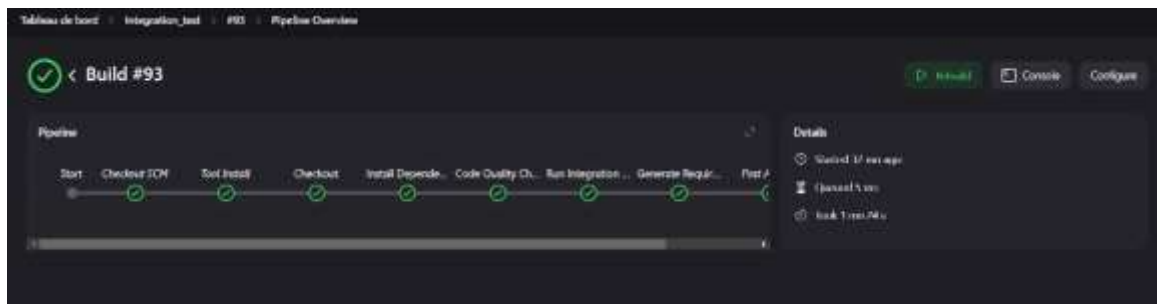
Mitigation Strategies

1. Use of Git and GitHub for version control and collaboration and WhatsApp for communication.
2. Use of AI tools and online resources to fill knowledge gaps.
3. Utilizing a member's Wi-Fi during group meetups.

Pipeline Metrics

The pipeline performance was measured by:

1. Ensuring all test jobs run properly for successful pipeline completion.



2. Generating a coverage report to measure the percentage of code base covered by unit test functions.

Coverage report: 76%

filter...

☐ hide covered

Files

Functions

Classes

coverage.py v7.6.8, created at 2025-01-05 09:53 +0100

File	statements	missing	excluded	coverage
app.py	4	4	0	0%
tests__init__.py	0	0	0	100%
tests\conftest.py	25	0	0	100%
tests\test_auth.py	27	0	0	100%
tests\test_views.py	36	0	0	100%
website__init__.py	29	1	0	97%
website\auth.py	53	10	0	81%
website\models.py	30	0	0	100%
website\views.py	83	54	0	35%
Total	287	69	0	76%

coverage.py v7.6.8, created at 2025-01-05 09:53 +0100

Test Procedure

Test Setup

Unit Test

1. The app to be tested and its necessary dependencies must be installed.
2. Flake8 must be installed to ensure proper code styling and formatting.
3. Pytest (for Python), JUnit, TestNG (for Java), Jest, Jasmine (for JavaScript), NUnit, and MSNet (for C#) must be properly installed.
4. The test cases should be written properly.

Execution Instructions

Unit Test

After writing the unit test case for a function, navigate to the parent directory containing the test file where this test case (test function) is found. Run the following command in your terminal:

```
bash
Copier
```

```
pytest test_filename.py
```

where filename is the name of your file. Ensure pytest is installed by checking its version:

```
bash
Copier
pytest --version
```

A test report will be displayed in your terminal after the command is written. Verify whether the function tested has passed. If it fails, review the test function again.



```
PS C:\Users\MM\Documents\MM First Semester Notes\Software Quality Tools\Software Testing\Software-Quality-Tools-Project> pytest tests
===== test session starts =====
platform: win32 -- python: 3.12.5, pytest: 6.3.4, pluggy: 1.5.0
rootdir: C:\Users\MM\Documents\MM First Semester Notes\Software Quality Tools\Software Testing\Software-Quality-Tools-Project
plugins: cov: 6.0.0
collected 11 items

tests/test_auth.py ..... [ 45%]
tests/test_views.py ..... [100%]

----- warnings summary -----
tests/test_auth.py::test_login_success
tests/test_auth.py::test_user_registration
C:\Users\MM\AppData\Local\Programs\Python\Python12\lib\site-packages\Flask_login\login_manager.py:488: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version, use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    expires = datetime.utcnow() + duration

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 11 passed, 2 warnings in 7.00s =====
PS C:\Users\MM\Documents\MM First Semester Notes\Software Quality Tools\Software Testing\Software-Quality-Tools-Project>
```

Integration Test

After creating and configuring the pipeline, run it in Jenkins. A build ID is generated after a pipeline is built. View the output in the pipeline output and status.

Requirements for Integration Test

1. All unit test cases should be working properly.
2. Jenkins should be installed and accessible through the browser.
3. The pipeline should be properly created and configured.

Expected Results

Unit Test


All unit test cases should return a passed status.

Résultats des tests : test_auth

0 échecs (±0)

5 tests (±0)

A pris 4,7 s.

 Ajouter une description

Tous les tests

Nom du test	Durée	Statut
test_login	0,12 s	En succès
test_login_invalid_password	0,1 s	En succès
test_login_success	2,9 s	En succès
test_logout	0,11 s	En succès
test_user_registration	1,4 s	En succès



Integration Test

The pipeline should return a status of "Success" when it has been built.

Error Handling

Common Errors

- Environment configuration issues.
- Dependency management issues.
- Test flakiness.
- Long running tests.
- Resource limitations.
- Jenkins configuration errors.

Mitigation Strategies

- Use Docker for consistent environments.
- Use requirements.txt for dependency management.
- Identify and stabilize flaky tests.
- Monitor and scale CI resources as needed.

- Regularly review Jenkins job configurations.
- Set coverage thresholds and use pytest-cov.

Test Report

Test Cases

- **test_login:** Ensures the user has signed up and checks credentials.
- **test_login_invalid_password:** Validates login with the correct password syntax.
- **test_login_success:** Checks if the login process was successful.
- **test_logout:** Tests the logout process.
- **test_user_registration:** Checks if the user is registered.
- **test_create_comment:** Verifies comment creation.
- **test_create_post:** Tests if the post creation aligns with the app parameters.
- **test_create_post_validation:** Tests the create_post_validation functionality.
- **test_delete_post:** Checks if a post has been deleted properly.
- **test_home_page:** Performs checks on the home page.
- **test_like_post:** Tests if a post was liked.

Results

Test Name	Duration	Status
test_login	0.1 s	success
test_login_invalid_password	96 ms	success
test_login_success	3.1 s	success
test_logout	0.1 s	success
test_user_registration	1.4 s	success
test_create_comment	90 ms	success
test_create_post	0.1 s	success
test_create_post_validation	88 ms	success
test_delete_post	0.1 s	success
test_home_page	0.16 s	success
test_like_post	96 ms	success

Summary

- **Tests Run:** 11
- **Tests Jumped:** 0
- **Tests Passed:** 11
- **Tests Failed:** 0
- **Overall:** 11 passed, 0 failed (100% passed).

IV. Discussion

The pipeline's setup ensures consistent automated testing, but the test coverage report highlights areas needing improvement. Addressing low-coverage files and securing the webhook configuration are critical for achieving robust CI practices. Future iterations should aim to secure webhook payloads and further expand test coverage.

V. Conclusion

This project demonstrated the setup and configuration of a CI pipeline using Jenkins for a GitHub-hosted project. The pipeline automates testing and provides valuable insights into code quality through coverage analysis. Continued efforts to enhance test coverage and secure configurations will solidify the project's reliability and success.

Individual contribution

S/N	NAMES	MATRICULE	Gitbub Name	Contribution
1	ARREY ABUNAW REGINA EBAI	FE22A152	GinaBlack	100%
2	ARREY-TABOT PASCALINE	FE22A151	Pascaline-sudo	95
3	ATANKEU TCHAKOUTE ANGE NATANAHEL	FE22A158	At-an	100%
4	ATIKU BORIS-BEN MFAI	FE22A159	-----	10%
5	AV'NYEHBEUHKOHNEZEKIEL HOSANA	FE22A160	AVEZEKIELHOSANA	95%
6	AWA ERIC ANGELO JUNIOR	FE22A162	ericBlack1	100%
7	AYUK NESTOR EYONG	FE22A164	themayor-23	85%
8	BEH CHU NELSON	FE22A170	CHUNELSON123	100%
9	BEKONO AKONO MARTHE LUNYA LAWRICA	FE22A171	lawrica123	95%
10	BELLAH LOVETTE MANYI	FE22A172	BELLAHLOVETTE	95%
11	KONINGS AKOKO	FE24A623	APKONINGS	90%