Continuous Integration Report

Cohort 1 Group 8

Roy Asiku, Tianqi Feng, Andrew Jenkins, Nicholas Lambert, Tom Byron

Summary of Continuous Integration Methods

Our project employs Continuous Integration (CI) through GitHub Actions to automate the merging and testing of code changes. This CI process is vital as it ensures the integrity of our codebase by automatically building and testing the project whenever changes are pushed to the remote repository. Our CI pipeline is triggered by any push to branches or tags and on pull requests to any branch, ensuring coverage of all changes before merging to the main branch.

The CI pipeline includes multiple jobs:

1. Build: Compiles the project across different operating systems (Ubuntu, Windows, macOS) and generates a coverage report.
2. Code Quality: Performs static code analysis to maintain high code standards.
3. Asset Validation: Checks the integrity of project assets.
4. Release: Manages the packaging and release of the software when tagged in the repository.

Continuous Integration Infrastructure

The workflow is defined in a YAML file within the. github/workflows directory of our repository, specifying each job and its steps. This file controls the execution of the entire CI process, from setup to completion of tasks like testing, analysis, and deployment.

Configuration Details:

- Trigger Conditions: The pipeline triggers on any push to branches or tags and on pull requests, which allows for immediate feedback on integration status.
- Build Job: This job is matrix-based, running on the latest versions of Ubuntu, Windows, and macOS. It involves setting up JDK 11, executing the Gradle build, and then uploading the JaCoCo coverage report and compiled JAR files as artefacts.
- Code-Quality Job: Runs only on Ubuntu, focusing on running static code analysis with Gradle to generate a check style report.
- Asset Validation Job: Verifies the integrity of assets using file type checks on Ubuntu.
- Release Job: Prepares and publishes artefacts when a new version is tagged, ensuring that releases are generated from the most stable and recent codebase.

Infrastructure Benefits:

- Cross-Platform Checks: Ensures that the application is tested across all major operating systems, which enhances compatibility.
- Automated Testing and Quality Assurance: Helps maintain a high standard of quality and functionality with every change.
- Release Automation: Simplifies the process of creating and distributing stable releases.