Continuous Integration

Group Name: Group 11

Group Number: 11

Group Members:

Zubair Shaik, Dom Decicco, Damian Myszko, Yousif Zuhair, Elijah Jones, Charlie Somerscales, Tawfig Osman

Our project leverages GitHub Actions for Continuous Integration, ensuring that any modifications to the code are continuously integrated, tested and maintained. The CI pipelines are structured to automate the building and testing process, therefore maintaining high code quality and minimising integration complications

Pipelines Overview:
Our project has two primary CI pipelines: **Build and Test Pipeline** and **Dependency Submission Pipeline**. Both pipelines are triggered by events in the GitHub Repository.

Pipeline Triggers:
1. Push to the ‘master’ branch: Whenever a commit is pushed to the ‘master’ branch of our project’s GitHub repository, the pipeline is triggered.
2. Pull Requests to the ‘master’ branch: The pipeline is also triggered by a pull request made to the ‘master’ branch. This allows for pre-merge testing, ensuring that new features or bug fixes are verified before merging into the ‘master’ branch.

Continuous Integration Pipelines
1. Build and Test Pipeline
Inputs:

- Source Code: The latest code from the ‘master’ branch or the pull request.
- Gradle Build Scripts: Gradle configuration files that define how the project should be built and tested.

Outputs:

- Build Artefacts: Compiled applications binaries and libraries.
- Test Reports: Results of the unit tests, including any test failures or errors.
- Coverage Reports: Code coverage data generated by JaCoCo, which shows how much of the code is covered by tests.

2. Dependency Submission Pipeline
Inputs:

- Dependency Graph Data: Information about the projects’ dependencies, generated by the Gradle dependency submission plugin.

Outputs:

- Dependency Graph: A dependency graph that can help to identify potential vulnerabilities and ensure that the project uses the latest version of dependencies.

Continuous Integration Pipelines Implementation

The CI Pipeline for our project is implemented using GitHub Actions and is defined in a YAML file located in the '.github/workflows' directory of the repository. The configuration includes two jobs: **'build'** and **'dependency-submission'**

Pipelines Breakdown

1. Build and Test Pipeline:

- **runs-on:** Specifies the environment in which the job will run as 'ubuntu-latest'.
- **permissions:** Sets read-only permissions for the job.
- **steps:** Lists the sequence of steps to be executed in the job.
    - **Checkout repository:** Uses 'actions/checkout@v4' action to checkout the latest code from the repository.
    - **Set up JDK:** Uses 'actions/setup-java@v4' to install and setup JDK 17.
    - **Cache Gradle Dependencies:** Uses 'actions/cache@v3' to cache Gradle dependencies to optimise build times.
    - **Build and Test Gradle wrapper:** Builds the project and runs the tests using the Gradle wrapper.

2. Dependency Submission Pipeline:

- **runs-on:** Specifies the environment in which the job will run as 'ubuntu-latest'.
- **permissions:** Grants write permission to the contents for dependency submission.
- **steps:** Lists the sequence of steps to be executed in the job.
    - **Checkout repository:** Uses 'actions/checkout@v4' action to checkout the latest code from the repository.
    - **Set up JDK:** Uses 'actions/setup-java@v4' to install and setup JDK 17.
    - **Generate and Submit Dependency graph:** Uses 'gradle/actions/dependency-submission@v3.1.0' to generate and submit the dependency graph.

The CI pipelines implemented in our project ensures continuous integration, maintaining code quality through the pre-defined automated builds and tests. This facilitates for seamless CI process, contributing to the reliability of our project.