Continuous Integration

Cohort 1 Group 1:
Ileri Adegeye
Maida Ahmed
Holly Ainsworth
Anthony Colin Jones
Luca Gilligan
Alex Joyner
Minhaj Zaidi

Continuous Integration

Continuous Integration (CI) is a software development practice in which code changes are frequently integrated into a shared version control repository and automatically validated through build and test processes. The primary goal of CI is to identify integration issues as early as possible, thereby reducing the risk of bugs accumulating over time.

The project follows a CI approach based on frequent, incremental commits to a shared Git repository. Each commit or pull request automatically triggers a CI workflow, which performs a sequence of predefined tasks including compiling the source code and executing automated tests. This ensures that all changes are continuously validated against the existing codebase, reducing the likelihood that new features or bug fixes introduce unintended errors. The use of automated workflow runs also provides immediate feedback on the success or failure of each change, allowing issues to be addressed promptly.

Given that the project is implemented in Java, the CI process is closely aligned with standard Java development practices. Gradle is used as the build automation tool, providing a reliable and reproducible way to compile the application, manage dependencies, and run test suites. Integrating Gradle into the CI pipeline ensures that the same build process is used both locally and within the automated environment, reducing discrepancies between development and integration environments. This consistency is essential for maintaining build reliability and avoiding “doesn’t work on my machine” issues.

Automated testing is a core component of the CI approach used in this project. Tests are executed as part of each workflow run, ensuring that core functionality is validated whenever code changes are introduced. This significantly reduces the risk of regressions, particularly as the codebase evolves and becomes more complex. By detecting failures early in the development lifecycle, CI minimizes the effort required to fix defects and helps maintain overall software quality.

This CI approach is particularly appropriate for the project due to its active development cycle and frequent modifications to core functionality. Even as a single-developer project, CI provides substantial benefits by enforcing disciplined development practices, improving code reliability, and increasing confidence in each code change. The use of automated Gradle-based builds and CI workflow runs ensures that the project remains stable, maintainable, and ready for further extension or deployment.

The use of continuous integration also streamlines teamwork by providing a shared, automated mechanism for validating contributions. By integrating changes frequently into a common repository and running CI workflows on every commit, all contributors receive immediate and consistent feedback on the impact of their changes.
This reduces integration conflicts, as issues are detected early rather than accumulating over time. The CI pipeline establishes a single source of truth for build and test results, improving transparency and communication within the team. As a result, developers can collaborate more efficiently, coordinate changes with greater confidence, and maintain a stable codebase even when multiple features or fixes are being developed concurrently.

Infrastructure

Our CI infrastructure for this project is built using industry-standard tools that support automation, reliability, and collaborative development. GitHub is used as the primary version control platform for the project. GitHub supports features such as commit histories, branching, and pull requests, helping maintain parallel development and code integrity. By using GitHub, all contributors work from a shared codebase, ensuring consistency and visibility of changes across the project.

GitHub Actions is used to implement continuous integration automation. All CI workflows are defined in the *.github/workflows* directory (as shown in figure 1). GitHub Actions allows CI workflows to be defined directly within the repository using YAML configuration files, enabling automated processes to run in response to code pushes or pull request submissions. Gradle is used as the build automation tool for the project. Gradle efficiently manages dependencies, compiles source code, and coordinates test execution, reducing discrepancies between local development and automated CI builds.

JUnit is used as the testing framework for the project. It enables the creation and execution of automated unit tests. JUnit tests are integrated into the build and are executed automatically during each CI workflow run. Unit tests are located within a dedicated test directory, allowing individual components to be tested independently. This separation improves test clarity and maintainability, while also ensuring that tests can be executed reliably within a headless CI environment. Together, these tools ensure that the project remains stable, maintainable, and scalable as development progresses.

The CI development follows a branch-based workflow, where a new branch is created whenever a new feature, enhancement, or fix is required, allowing for changes to be made in isolation without affecting the main code base. Github Issues are used to document and manage specific problems encountered during development, such as build failures, test errors and implementation challenges. Each run begins with a job setup phase, where the execution environment is prepared. The pipeline then retrieves the latest version code from the repository and ensures the build operates the exact code state when triggering a commit or pull request.

Following code checkout, the pipeline proceeds with Gradle configuration and execution.The project is then built using the Gradle wrapper, which compiles the source code, resolves dependencies, and executes defined build tasks. This ensures that the same Gradle version is used across all environments, improving build reproducibility and ensures that all tests are consistently executed whenever the pipeline runs.

Evidence of Use

Figure 1: Github Project Overview with relevant parts highlighted (https://eng1-c1g1.github.io/assets/images/CI/figure_1.png)
Figure 2: Successful build and its operations (https://eng1-c1g1.github.io/assets/images/CI/figure_2.png)
Figure 3: JUnit tests directory (https://eng1-c1g1.github.io/assets/images/CI/figure_3.png)
Figure 4: JUnit tests Workflow (https://eng1-c1g1.github.io/assets/images/CI/figure_4.png)
Figure 5: JUnit tests artifact with the test report (https://eng1-c1g1.github.io/assets/images/CI/figure_5.png)