CI Pipeline

Design of CI Pipeline

There are three main stages in a CI Pipeline (CI/CD Process, n.d.):

1. Source
2. Build
3. Test
    - Feedback

I will not cover the fourth stage 'Deploy' as this project only deals with a simulated drone for the purpose of an assignment. However, should the scope of the project increase and be suitable for deployment then a full CI/CD Pipeline would need to be implemented.

Source

The pipeline will be triggered on every push and pull request to the GitHub repository to ensure all changes are verified before merging or being pushed.

Build

In this stage, a Docker image would be built using a Dockerfile. My system so far has been built with IntelliJ's inbuilt Maven tools which my CI pipeline would be integrated into this so it could use the tools already available to conduct a static code analysis. This could include Maven's default checks or IntelliJ's compiler warnings, ensuring the build fails on configuration or compilation errors. In this stage, any preliminary unit tests would be run and, in my system, this includes the geometric and navigational tests for drone movement. This adds to code quality assurance. Any code compilation or configuration errors are caught at this stage.

Test

In this stage, after the code has compiled successfully, all automated tests are run. For my system this includes the integration tests for order validation and REST endpoints. To ensure that low level errors are caught early, these tests will be run in a bottom-up manner (Unit -> Integration) which this pipeline allows for by running the unit tests in the 'Build' stage. Therefore, the pipeline obeys the principle of 'fail fast' (CI/CD Process, n.d.). Performance tests would also be included to ensure that changes do not introduce unintended regressions in response time, particularly for flight path generation.

Functional and security tests were not needed for the scope of the ILP coursework however would be necessary to implement if the system were to be deployed. This would include user interface tests and tests to identify security vulnerabilities.

Feedback

After testing is complete, test results are immediately available. This includes a report on test coverage and functionality.

Embedding of Testing

One of the primary objectives of a CI pipeline is for there to be little to no need for a developer to manually get involved in the testing and deployment process after committing the changes they've made unless an error occurs. This requires the full automation of the testing suite.

The majority of the testing I've conducted in this project is suitable for full automation and would be embedded directly into the pipeline.

The automated tests would be:

- Order validation tests covering all functional requirements and boundary conditions
- Drone movement and geometric correctness tests
- REST-level integration tests verifying HTTP behaviour and response structure

System level tests require the manual inspection of paths through GeoJson and therefore are excluded from the pipeline.

Issue Detection

The following issues would be identified by the CI pipeline:

Kind of issue	Examples and Detection
Incorrect validation logic	Missing delivery fees, Unidentified pizza or restaurant These would cause the integration tests to fail in the 'Test' stage due to failing the assertions in the test suite.
Boundary condition violations	Exceeding 4 pizzas in one order, Incorrect expiry date handling These would cause the integration tests to fail in the 'Test' stage due to failing the assertions in the test suite.
Reductions in test coverage and/or performance levels	This would be identified at the end in the feedback reports.
Empty or malformed docker image	This would be caught in the 'Build' stage when the docker image is constructed. This will prevent the pipeline from moving on to the next stage and it will fail.

Bibliography

CI/CD Process. (n.d.). Retrieved from CodeFresh: https://codefresh.io/learn/ci-cd-pipelines/ci-cd-process-flow-stages-and-critical-best-practices/