LO5: Code Review and CI Pipeline for PizzaDronz Project

Introduction

This document describes the code review techniques that have been used and could have been used given more time-resource. The review results are also discussed. It also describes construction of a Continuous Integration (CI) pipeline for the project and what test automation could have been done. For the latter, it is assumed that the DevOps lifecycle is used.

Review Techniques and Results

The main code review technique, which was extensively used in the development process was automated code review. It was done by Intellij IDEA, e.g. error highlighting and warnings. Manual review was supplementary but still very useful. If the review was to be carried out by a different person, i.e. an assigned reviewer, then code and test distribution should first take place, followed by an "over-the-shoulder" review, a discussion of what the code is expected to do and an explicit review report.

Code style was only maintained manually, but it could have been automated by specifying certain style rules (as it is possible to do in such version control systems as Bitbucket or Git). Then if a piece of code does not satisfy those rules, either the build would fail or it would at least be indicated and thus could easily be fixed.
As a result of manual and automated code review, a big amount of errors were fixed and potentially dangerous statements (unchecked casting, statements producing exceptions, etc.) were eliminated.

CI Pipeline Construction

If we were to build a CI pipeline, it would have the following stage-by-stage structure:

1. Source Stage
   The IDE where Java code is developed is Intellij IDEA. The version control system is Git, and the code is stored in a shared repository (in case of multiple developers) on GitHub. Thus the trigger for the build stage would be a Git push.

2. Build Stage
   The build automation tool used for this project is Maven. For our project, it is a convenient tool which combines together source code from the repository and links it with the necessary libraries (all required artifacts are stored in the repository). The build has to pass initial tests in order to be successful and to progress to the next stage of the pipeline. If the build fails, corrections must be made and the build re-run. Container and dependency scanning would also take place where applicable.

3. Test Stage
   There are a number of testing activities a pipeline would undertake, these will be described in detail in the next section.

4. Deploy Stage
   First alpha-testing would be carried out with a dedicated test server and simulation of real databases. Then beta-testing would take place with the new build deployed in parallel to the current latest one. Once it is properly functioning, it totally replaces the old build for all users. A very popular CI engine Jenkins could be used for the deploy stage.

Test Automation (Test Stage of the Pipeline)

The following aspects of the testing process could be automated in the CI pipeline:

- Unit testing – would ensure correctness of any new features of the build.
- Integration testing – it should be ensured component which has been modified functions properly when operating with other components of the system.
- Regression testing – it should be verified that the new alterations do not violate any previously finished features, i.e. there is no regression of the system.
- Performance testing – it should be ensured the new build can operate under certain conditions.

For out project, Selenium automation software could be used, as the main focus is on the functional tests, for which Selenium is one of the most suitable automation tools.

CI Pipeline Functions

The pipeline would be able to identify the following issues:

- At Build Stage:
  - ➢ If the code is functionally correct (if the build is successful)
  - ➢ If all initial tests pass (if not, we do not proceed to the next stage and have to fix the build)
- At Test Stage:
  - ➢ If all unit and integration tests pass (total functional correctness)
  - ➢ If the new build does not break any previous correct functionality (regression)
  - ➢ If the system can (still) operate under significant load.