QA Manual

Version 2.3.1

Overleaf

June 23, 2022

Contents

List of Code Listings

1 Project Management

1.1 Team and Roles

Role	Member
Team Coordinator	Tomé Carvalho
Product Owner	Lucius Vinicius
QA Engineer	Dinis Lei
DevOps Master	Camila Fonseca
Developer	All

Table 1: Team and Roles

1.2 Agile Backlog Management and Assignment

This projects uses Jira's Backlog to define and manage issues, stories and tasks. Each repository has its own backlog column and every issue created related to the repository should go the respective column.

The creator of the issue should assign a estimate of points worth, for reference, 1 point is roughly equivalent to 1 hour. If the issue is done faster or slower than the foreseen time, its points should be updated to reflect the real time.

2 Code Quality Management

2.1 Guidelines for contributors

Coding style will mostly follow google's java coding style
(source: https://google.github.io/styleguide/javaguide.html)

2.2 Code quality metrics

Static code analysis will be done using SonarQube and JacCoCo for code coverage. If the code does meet the quality gate expectations it should be reformulated.

Quality Gate defined is:

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Figure 1: Quality Gate Metrics

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

This project adopts git workflow, that means it uses feature branching to develop new features.

To associate commits, pull requests and branches to issues, the id should be added as a prefix, for example:

```
git checkout -b TQSG11-5-new-branch

git commit -m "TQSG11-5 <message>"
```

3.2 CI/CD pipeline and tools

The CI/CD pipeline is made using Github's Actions. For every pull request and push to the 'main' and 'dev' branches, Github's Actions runs the tests, builds the project and runs Sonar Qube analysis. Pull requests for the 'main' branch are blocked if any of the actions fails, but for develop it is still possible, but not recommended, to merge the pull request.

Github's Actions

4 Software testing

4.1 Overall strategy for testing

The projects uses a TDD (Test Driven Development) approach of the APIs and A BDD (Behaviour Driven Development) with Cucumber for both Frontends.

4.2 Functional testing/acceptance

The strategy adopted to write Functional Tests is closed box, the developer should make the tests based on the agreed upon functional requirements, without the need to know how the inner code will work.

4.3 Unit tests

The strategy adopted to write Unit Tests is open box, the developer should have already an idea of how the inner workings of the functions should behave even if it hasn't been implemented yet

4.4 System and integration testing

The strategy adopted to write Integration Tests is closed box, the tester doesn't need to know how the whole API works just the desired input and output.