TQS: Quality Assurance manual

Alexandre Gazur (102751)

Daniel Ferreira (102885)

Emanuel Marques (102565)

Ricardo Pinto (103078)

v16-05-2023

[**1**](#_gjdgxs) **Project management 1**

[1.1 Team and roles 1](#_30j0zll)

[1.2 Agile backlog management and work assignment 1](#_1fob9te)

[**2**](#_3znysh7) **Code quality management 2**

[2.1 Guidelines for contributors (coding style) 2](#_2et92p0)

[2.2 Code quality metrics 2](#_tyjcwt)

[**3**](#_3dy6vkm) **Continuous delivery pipeline (CI/CD) 2**

[3.1 Development workflow 2](#_1t3h5sf)

[3.2 CI/CD pipeline and tools 2](#_2s8eyo1)

[3.3 System observability 2](#_17dp8vu)

[3.4 Artifacts repository [Optional] 2](#_3rdcrjn)

[**4**](#_26in1rg) **Software testing 2**

[4.1 Overall strategy for testing 2](#_lnxbz9)

[4.2 Functional testing/acceptance 3](#_35nkun2)

[4.3 Unit tests 3](#_1ksv4uv)

[4.4 System and integration testing 3](#_44sinio)

[4.5 Performance testing [Optional] 3](#_2jxsxqh)

# Project management

## Team and roles

Alexandre Gazur - Team Leader, PickUs backend

Ricardo Pinto - QA Engineer, PickUs backend and QA

Daniel Ferreira - Product Owner, Web eStore

Emanuel - DevOps Master, PickUs frontend (admin dashboard and ACP dashboard)

## Agile backlog management and work assignment

User stories are defined in [our Jira backlog](https://grupo-ies.atlassian.net/jira/software/projects/TQSPROJ/boards/4/backlog). They may be added, updated or removed as things change (agile). A few of the not implemented stories are prioritized every iteration (1 week). They are then implemented, and if they pass the tests, the pull request is accepted.

# Code quality management

## Guidelines for contributors (coding style)

In general, we follow the [AOSP coding style](https://source.android.com/docs/setup/contribute/code-style), since we think it makes a lot of sense.

## Code quality metrics

For static code analysis, we use SonarQube. All pull requests must pass the quality gate to be accepted, as well as all the tests.

We follow the [Clean as you Code](https://docs.sonarqube.org/latest/user-guide/clean-as-you-code/) paradigm to ensure minimal technical debt is added every iteration.

Some conditions of our quality gate are: more than 75% code coverage is required, no new blocker issues, no bugs and a maximum of 1 hour of technical debt for a new feature.

# Continuous delivery pipeline (CI/CD)

## Development workflow

We follow the [Github Flow](https://docs.github.com/en/get-started/quickstart/github-flow), because it is simple and effective. Developers choose a prioritized user story from [our Jira backlog](https://grupo-ies.atlassian.net/jira/software/projects/TQSPROJ/boards/4/backlog), make a branch for it, implement it, then make a pull request to the main branch.

As for refactoring, some is done at the end of the iteration, and the rest is done in the last 2 iterations (18/05/2023 - 31/05/2023).

Definition of done:

the user story is doable in our system;

the story has tests associated and all of them are passed;

the frontend, backend and database are connected and adapted to each other, they communicate/interact as expected.

## CI/CD pipeline and tools

In every pull request, a github workflow (defined with a yml file in the repo) runs the tests. We also use SonarCloud for CI: in every pull request, SonarCloud’s workflow will perform a static code analysis. All the tests and the quality gate must be passed in order for the PR to be accepted (may be accepted/refused automatically or manually).

# Software testing

## Overall strategy for testing

We use a mix of 2 strategies: Test After Development (TAD) and Test Driven Development (TDD).

In TAD, we first implement the features and then write tests for it. We chose this strategy because the team members are used to it and have a lot of experience using it. Not only that, we are not very sure of the specific Java code architecture (classes, functions, …) at this time, so it becomes hard to use Test Driven Development.

When the QA Engineer has time to work on the project, but features are not yet implemented, or when we know clearly how a part of the Java code architecture would look like, TDD is used.

We use tools like REST Assured, JUnit and Selenium IDE.

## Functional testing/acceptance

For functional testing, we test the system by simulating how an user would use it. If it behaves as expected, the test is passed. If something is off during the simulation (e.g. a table in the frontend not displaying the expected items), it is corrected and then the test is reran.

Useful tools for functional testing include Selenium IDE.

## Unit tests

For unit testing, we use a mix of open box testing and developer perspective testing.

In open box, tests are written during development; this is useful, for example, to make sure a complex Java function is working properly.

For simpler logic, we use developer perspective testing, where tests are written after development. For example, when a developer writes a simple Java function, he may know that, very probably, the function works as expected; tests for this function can be postponed so the developer can focus on implementing the feature.

Useful tools for unit testing include JUnit.

## System and integration testing

For integration testing, such as the API, we use black box testing.

For example, we execute a GET call, maybe with some URL parameters, and verify that the returned response is as expected. Or, we execute a POST call, and confirm that the database is updated as expected.

Useful tools for integration testing include test containers and REST Assured.