TQS: Quality Assurance manual

***Flávia Figueiredo [88887]*, *Pedro Marques [89069], José Frias [89206], Tomás Batista [89296]***

v2020-06-04

[**1**](#_heading=h.gjdgxs) **Project management 1**

[1.1](#_heading=h.30j0zll) Team and roles 1

[1.2](#_heading=h.1fob9te) Agile backlog management and work assignment 1

[**2**](#_heading=h.3znysh7) **Code quality management 2**

[2.1](#_heading=h.2et92p0) Guidelines for contributors (coding style) 2

[2.2](#_heading=h.tyjcwt) Code quality metrics 2

[**3**](#_heading=h.3dy6vkm) **Continuous delivery pipeline (CI/CD) 2**

[3.1](#_heading=h.1t3h5sf) Development workflow 2

[3.2](#_heading=h.2s8eyo1) CI/CD pipeline and tools 2

[3.3](#_heading=h.17dp8vu) Artifacts repository [Optional] 2

[**4**](#_heading=h.3rdcrjn) **Software testing 2**

[1.1](#_heading=h.26in1rg) Overall strategy for testing 2

[1.](#_heading=h.lnxbz9) Functional testing/acceptance 2

[2.](#_heading=h.35nkun2) Unit tests 3

[3.](#_heading=h.1ksv4uv) System and integration testing 3

# Project management

## Team and roles

Tomás Batista is the Team Manager, responsible of making a fair distribution of the work between the team, dealing also with the project calendar, relatively to the outcomes that must be delivered in each iteration.

Flávia Figueiredo is the Product Owner, responsible of representing the stakeholders, giving orientations to the team about how a certain functionality should behave, since it has the best overview of the product.

Pedro Marques is the DevOps Master, responsible to prepare the deployment of the product, as well the setup of the git repository, configuring the development and production infrastructure.

José Frias is the Architect, responsible for designing the database and the endpoints. Also defined the testing strategy.

All members contribute to the development process of the system.

## Agile backlog management and work assignment

For the backlog management, the Pivotal Tracker workflow appears, where a user story is created and prioritized. After it is started, and when the development is concluded tests are needed to verify the increment using CI. Next, the stories need to be accepted by the Product Owner, moving that story to the Done panel after it is reviewed and accepted.

Concerning the agile practices, in each iteration a brief meeting was done to present the work done that iteration and the work assignment and planning for the next iteration.

The work was assigned using Trello, mainly making the division and assign the jobs between frontend and backend.

# Code quality management

## Guidelines for contributors (coding style)

Relatively to the coding style, the standard coding style for Java was adopted, concerning the naming of classes and variables and the naming of methods in order to be concise and readable. Moreover, the exception treatment was also considered.

We also added two things: one file, CONTRIBUTING.md, which specified the pull request process and the code of condut and also added issues templates for the users to fill when they discover a bug on our project, so the issue can be well detailed.

## Code quality metrics

For the static code analysis, two approaches were considered: the first, considering a local solution, where the IDE tools were considered, using the code inspection available in Intellij IDE); the second approach considered was Sonarcloud, used as a cloud and local solution, that is, before the code was committed or merged, concerning the backend, a manual verification with Sonarcloud was done locally. After the quality gates were met, the new code could be pushed. When that new code was merged, the Continuous Integration methods automatically contact with the sonarcloud to verify again that code against the quality gates. The quality gates defined were A at Maintainability, Security and Reliability. For the code coverage, a minimum of 80% was considered and for the duplication blocks no more than 3%.

We also added Codacy to give us quick feedback on the quality code. Both of the platforms (SonarCloud and Codacy have badges on README.md to give the developers quick view of the quality)

# Continuous delivery pipeline (CI/CD)

## Development workflow

Relatively to the workflow in github, each user story presented in Pivotal Tracker corresponds to a new branch in the repository, following the GitHub flow. After, when the development of the code is finished a pull request is originated, verifying the code against the quality gates of sonarcloud.

The “Definition of Done” of the team relatively to the user stories is that the tests passed, whether that be unit tests or integration tests, the Product Owner accepted the story, the endpoint is well defined and the acceptance criteria is present in the feature implementation. Moreover, the new code must pass the quality gates defined. The user interface was not considered for the definition of done.

## CI/CD pipeline and tools

The CI pipeline, using Github Actions, was to verify the increments and the features added to the master, whether that be through pull requests or pushes, where the tests were run and the build was verified. All the tests must pass, whether that be integration or unit tests. Sonarcloud was also considered during the CI pipeline, to verify the quality gates of the new code.

The VM is responsible to run the CD pipeline, generating a new artifact of the product, deploying it using a docker container.

## Artifacts repository

The artifacts generated during the Continuous Deployment are stored in the VM device, since they are then used to produce docker containers.

# Software testing

## Overall strategy for testing

The overall test development strategy was TDD, where for a specific functionality, tests were first written (seeing them fail) and only after the code was developed, making the tests pass. After that, refactoring for the new implemented feature could be done, moving then to the next functionality to be implemented, applying the same strategy.

## Functional testing/acceptance

For the functional testing, the tool used was Selenium Webdriver to test the user interface and the path flow between the different pages.

Closed box was the policy, since the backend was abstracted in the interface.

## Unit tests

Since Spring is based in a layered architecture, unit tests were done to verify each layer, that is, unit tests for the repository, service and controller.

The policy adopted was open box, since the tests were done taking into account the internal structure of the product, where the tester was also familiar with the code produced, seeing if the requirements were met relatively to the projected outcomes of a particular functionality, with the advantage of finding errors in an early stage of the development.

## System and integration testing

For the API testing REST Assured was used, firstly considering only the controller unit tests and then considering the integration testing.

This tests followed mainly the closed box approach, since the different endpoints were tested integrated with the repository and the internal structure was abstracted since the endpoints were called and records were inserted in the repository, but the flow between the controller and the repository was abstracted.