TQS: Quality Assurance manual

***André Morais [93236], Eduardo Santos [93107], Pedro Bastos [93150], Pedro Santos [93221]***

v2021-05-30

[1 Project management 1](#_Toc39440117)

[1.1 Team and roles 1](#_Toc39440118)

[1.2 Agile backlog management and work assignment 1](#_Toc39440119)

[2 Code quality management 2](#_Toc39440120)

[2.1 Guidelines for contributors (coding style) 2](#_Toc39440121)

[2.2 Code quality metrics 2](#_Toc39440122)

[3 Continuous delivery pipeline (CI/CD) 2](#_Toc39440123)

[3.1 Development workflow 2](#_Toc39440124)

[3.2 CI/CD pipeline and tools 2](#_Toc39440125)

[3.3 Artifacts repository [Optional] 2](#_Toc39440126)

[4 Software testing 2](#_Toc39440127)

[1.1 Overall strategy for testing 2](#_Toc39440128)

[1. Functional testing/acceptance 2](#_Toc39440129)

[2. Unit tests 3](#_Toc39440130)

[3. System and integration testing 3](#_Toc39440131)

[4. Performance testing [Optional] 3](#_Toc39440132)

[This report should be written for new members coming to the project and needing to learn what are the QA practices defined. Provide concise, but informative content, allowing other software engineers to understand the practices and quickly access the resources.

Tips on the expected content, along the document, are meant to be removed.

You may use English or Portuguese; do not mix.]

# Project management

## Team and roles

**Team Coordinator – *Pedro Bastos***

Tasked with fairly coordinating the team and fairly distributing tasks amongst its members. Taking the initiative solve problems that may appear and doing regular checkups to ensure everyone and everything is working according to plan. Must also make sure the project’s expected outcomes are being delivered in time.

**Product Owner – *Pedro Santos***

Must have a deep understanding of the problem presented, successfully representing the interests of the stakeholders. Should be able to answer questions that may arise amongst the team members relative to the expected features that the system must have. Is also involved in accepting solution increments making sure the correct functionalities are delivered and working as intended.

**QA Engineer – *André Morais***

Responsible with ensuring the quality of the product being developed, by promoting quality assurance practices and ensuring they are being used, and by utilizing the tools available to measure the quality of the project and its increments.

**DevOps Master – *Eduardo Santos***

In charge of the development and production infrastructure and its required configurations, ensuring it works correctly. Should also initially prepare the deployment machines/ containers ensuring they are ready for deployment, create and prepare the git repository so all members have access and can begin coding, and prepare other needed technologies such as a cloud infrastructures or database operations.

**Developer – *All***

All members will contribute to the development of the project’s tasks.

## Agile backlog management and work assignment

For the backlog management and work assignment we’ll be utilizing [GitHub’s Project Management](https://github.com/features/project-management/) features.

In the projects section we’ll include 2 projects, one for the Business Initiative (Covid Tests Deliveries) and one for the Deliveries Engine, each one containing their own tasks based on user stories separated into “To Do”, “In Progress” and “Done” columns.

When beginning work on a user story, a developer must first create an issue (which can represent features, bugs, and other development tasks), which will represent the story being worked on. Issues have a small title, a description detailing what needs to be done, labels to easily identify what category they belong to, a specific project which they belong to, and can be assigned to one or more team members, therefore facilitating job assignment. When all work is complete, the issue should be marked as closed.

# Code quality management

## Guidelines for contributors (coding style)

For this project we’ll be following the coding style [AOSP Java Code Style](https://source.android.com/setup/contribute/code-style).

## Code quality metrics

The main tool we’ll be utilizing for static code analysis will be Sonar Cloud, which will analyze our projects on every commit and pull request with a successful pipeline on GitHub.

The quality gate will be the built-in quality gate, recommended by Sonar Cloud, which focuses on keeping new code clean, to minimize the project’s debt time and therefore lowering the amount of effort needed to improve old code.

# Continuous delivery pipeline (CI/CD)

## Development workflow

To facilitate the development of new features we’ll be utilizing the [Git Feature Branch Workflow](https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow), with a main branch that will be updated whenever a release is ready, and a develop branch which will include the changes made during product development. Whenever a release is due, a pull request to merge develop into master will be made, and group members will make sure everything is working as expected before the pull request is accepted.

When contributing to the develop branch, developers must create a branch which will map to an open issue, after the necessary changes are made and the issue is complete, a pull request must be made to merge the feature branch into develop, this request should be reviewed by at least one other assigned member to make sure the new functionality is working as expected and the new code follows the project coding style guidelines.

As stated above, a feature branch must map to an open issue, which maps to a user story on the project’s backlog.

A user story is considered done when all functionalities needed for it to work have been developed, it has been subject to testing (both static as well as by using the system) and these were successful, the code has been reviewed and the system with its new changes is ready to be used by the target users.

## CI/CD pipeline and tools

To implement Continuous Integration into our project we’ll be using [GitHub Actions](https://github.com/features/actions), which provides an easy way to build pipelines and run them when certain conditions are meant.

We’ll be creating scripts which run on every push and pull request, each script will belong to a project. These scripts will build the project and run the available tests, if those phases are successful, the project it belongs to will then be analyzed, and the results will be available on the Sonar Cloud platform.

For the **CI** part, there are two scripts, one for each project, both of them will run on the latest ubuntu image.

* ***deliveries-engine.yml***: this worflow will trigger 2 jobs:
  + **build** - build the deliveries-engine Maven project, checking if it succeeds
  + **sonar** - runs the SonarCloud static code analysis for the deliveries-engine project, checking if it passes the quality gate
* ***t-tracker.yml***: this worflow will trigger two jobs:
  + **build** - build the t-tracker Maven project, checking if it succeeds
  + **sonar** - runs the SonarCloud static code analysis for the t-tracker project, checking if it passes the quality gate

For the **CD** part, there are also two scripts, one for each project, but, in this case, each one of them will run on a [GitHub Self-Hosted Runner](https://docs.github.com/en/actions/hosting-your-own-runners/about-self-hosted-runners) installed on the given virtual machine.

These runners allow us to easily deploy our projects, when pushing/pull requesting to specific branches.

Each project has its own ***docker-compose.yml***, which will deploy all the parts of the same project, this parts being:

* **SpringBoot project**
* **MySQL Database**
* **Frontend**

There was an issue with the ***sudo*** command, as the runners were running on one of the group members' user, and this depends on the UA's IDP, this could give permissions/timeouts errors, when executing the ***sudo docker-compose up*** command.

Background pattern

Description automatically generatedTo get around this, I created a new user, giving him root permissions. This can be done with the following commands:

After this step, we just have to enter the **VM** through the from the new user through **SSH**, with the previously defined password:

And install the runners through this new user. This prevented the previous errors from ocurring.

## Artifacts repository [Optional]

[Description of the practices defined in the project for local management of Maven *artifacts* and associated resources. E.g.: a[rtifactory](https://www.jfrog.com/artifactory/)]

# Software testing

## Overall strategy for testing

For testing of our backend functionalities, we’re planning to use TDD, writing tests for the feature we are developing before developing the actual methods and logic needed.

To test our user interface, we’re choosing to use BDD, integrating Cucumber with Selenium, so that our tests represent real scenarios.

## Functional testing/acceptance

[Project policy for writing functional tests (closed box, user perspective) and associated resources.]

## Unit tests

[Project policy for writing unit tests (open box, developer perspective) and associated resources.]

## System and integration testing

[Project policy for writing integration tests (open or closed box, developer perspective) and associated resources.]

API testing

## Performance testing [Optional]

[Project policy for writing performance tests and associated resources.]