TQS: Quality Assurance manual

***Grupo 105***

v0000-00-00

[1.1 Team and roles 1](#_Toc40157973)

[1.2 Agile backlog management and work assignment 1](#_Toc40157974)

[2.1 Guidelines for contributors (coding style) 2](#_Toc40157975)

[2.2 Code quality metric 2](#_Toc40157976)

[2.3 Git Standards 2](#_Toc40157977)

[3.1 Development workflow 3](#_Toc40157978)

[3.2 CI/CD pipeline and tools 3](#_Toc40157979)

[3.3 Artifacts repository [Optional] 3](#_Toc40157980)

[1.1 Overall strategy for testing 3](#_Toc40157981)

[1. Functional testing/acceptance 4](#_Toc40157982)

[2. Unit tests 4](#_Toc40157983)

[3. System and integration testing 4](#_Toc40157984)

[4. Performance testing [Optional] 4](#_Toc40157985)

# Project management

## Team and roles

**Team manager**: Tomás Costa

**DevOps master**: João Marques

**Product owner**: Francisco Jesus

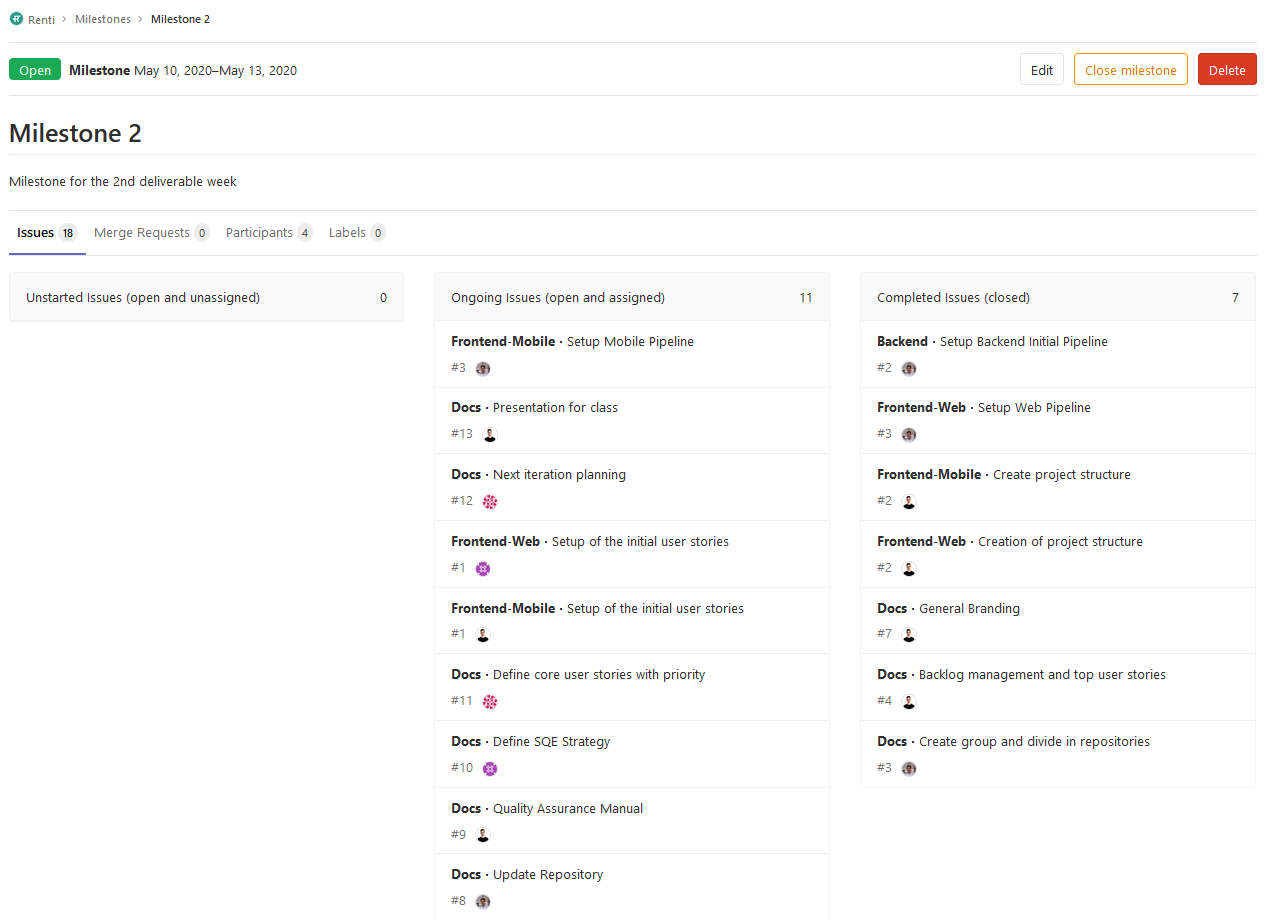
**Developer**: Everyone

**Backend:** João Marques, Francisco Jesus

**Frontend:** Tomás Costa, Miguel Matos

## Agile backlog management and work assignment

For backlog management we are using GitLab Boards and Milestones, assigning each task to a specific developer. We are experiencing GitLab instead of Jira, to expand our technology stack, and see how well it works.



# Code quality management

## Guidelines for contributors (coding style)

Still a work in progress, but heavily inspired in [AOS project](https://source.android.com/source/code-style.html) with standards like:

* + Dont ignore exceptions
  + Dont catch generic exceptions
  + Defining fields in standard places
  + Using TODO Comments
  + Logging instead of printing
  + Using standard bracket style

Also some standards from “[Clean Code](https://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882)”:

* + Avoid duplication anywhere in code
  + [Law of Demeter](https://dzone.com/articles/the-genius-of-the-law-of-demeter)

## Code quality metric

[Description of practices defined in the project for *static code analysis* and associated resources.]

[Which quality gates were defined? What was the r[ationale?]

Still a work in progress, check [link](https://www.perforce.com/blog/sca/what-static-analysis).

Expected use of SonarQube for static code analysis, due to past experience.

## Git Standards

GitLab was the obvious choice for the Git Platform since it has easier CI/CD Integration and our backlog management, which allows us to close tasks in commits. Some standards are:

* + Never merge directly, always make pull requests and identify at least one person to check (review) that pull request before merging the PR. (All repositories are configured to not accept a single person merge)
  + **New feature branch:** For each new feature create a branch following the standard: feature/<feature\_name>.
  + **New Issue branch:** For each fix create a branch following the standard: fix/<fix-name>.
  + Closing issues/tasks can be done by writing in commit message: “this closes #<issue\_nr>”

# Continuous delivery pipeline (CI/CD)

## Development workflow

[Clarify the workflow adopted [e.g.. [gitflow](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) workflow, [github flow](https://guides.github.com/introduction/flow/) . How do they map to the user stories?]

[Description of the practices defined in the project for *code review* and associated resources.]

[What is your team “[Definition of done](https://medium.com/@anca_51481/user-story-definition-of-done-dod-in-agile-software-development-and-the-technical-debt-a3abf6821ef2)” for a user story?]

## CI/CD pipeline and tools

[Description of the practices defined in the project for the continuous integration of increments and associated resources. Provide details on the tools setup and config.]

[Description of practices for continuous delivery, likely to be based on *containers*]

## 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

* 1. Overall strategy for testing

## Overall strategy for testing

Still a work in progress, check “Clean Code” chapters for testing and these links:

* + [Strategies](https://www.tutorialride.com/software-engineering/strategies-of-software-testing.htm)
  + [Testing overview](https://www.tatvasoft.com/blog/software-testing-quality-assurance-strategies/)

[what was the overall test development strategy? E.g.: did you do TDD? Did you choose to use Cucumber and BDD? Did you mix different testing tools, like REST-Assured and Cucumber?...]

**Three Laws of TDD**

* + You may not write production code until you have written a failing unit test.
  + You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
  + You may not write more production code than is sufficient to pass the currently failing test.

## 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.]