

TQS: Quality Assurance manual

Grupo 105

v2020-05-12

1.1	Team and roles.....	1
1.2	Agile backlog management and work assignment	1
2.1	Guidelines for contributors (coding style)	2
2.2	Code quality metric.....	2
2.3	Git Standards.....	2
3.1	Development workflow	3
3.2	CI/CD pipeline and tools	3
3.3	Artifacts repository [Optional].....	3
1.1	Overall strategy for testing.....	3
1.	Functional testing/acceptance	4
2.	Unit tests.....	4
3.	System and integration testing.....	4
4.	Performance testing [Optional]	4

1 Project management

1.1 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

1.2 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.

Renti > Milestones > Milestone 2

Open Milestone May 10, 2020–May 13, 2020 Edit Close milestone Delete

Milestone 2

Milestone for the 2nd deliverable week

Issues 18 Merge Requests 0 Participants 4 Labels 0

Unstarted Issues (open and unassigned)	Ongoing Issues (open and assigned)	Completed Issues (closed)
0	11	7
	Frontend-Mobile · Setup Mobile Pipeline #3	Backend · Setup Backend Initial Pipeline #2
	Docs · Presentation for class #13	Frontend-Web · Setup Web Pipeline #3
	Docs · Next iteration planning #12	Frontend-Mobile · Create project structure #2
	Frontend-Web · Setup of the initial user stories #1	Frontend-Web · Creation of project structure #2
	Frontend-Mobile · Setup of the initial user stories #1	Docs · General Branding #7
	Docs · Define core user stories with priority #11	Docs · Backlog management and top user stories #4
	Docs · Define SQE Strategy #10	Docs · Create group and divide in repositories #3
	Docs · Quality Assurance Manual #9	
	Docs · Update Repository #8	

2 Code quality management

2.1 Guidelines for contributors (coding style)

Still a work in progress, but heavily inspired in [AOS project](#) 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](#)”:

- Avoid duplication anywhere in code
- [Law of Demeter](#)

2.2 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 rationale?]

Still a work in progress, check [link](#).

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

2.3 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>”

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

[Clarify the workflow adopted [e.g.. [gitflow](#) workflow, [github 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](#)” for a user story?]

3.2 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*]

3.3 Artifacts repository [Optional]

[Description of the practices defined in the project for local management of Maven *artifacts* and associated resources. E.g.: [artifactory](#)]

4 Software testing

4.1 Overall strategy for testing

1.1 Overall strategy for testing

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

- [Strategies](#)
- [Testing overview](#)

[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.

1. Functional testing/acceptance

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

2. Unit tests

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

3. System and integration testing

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

API testing

4. Performance testing [Optional]

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