TQS: Quality Assurance manual

Diogo Carvalho [92969], Pedro Amaral [93283], Rafael Baptista [93367], Ricardo Cruz [93118]

[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.]

1 Project management

1.1 Team and roles

Members	Roles	Responsibilities
Diogo Carvalho	Team Coordinator	“Ensure that there is a fair distribution of tasks and that members work according to the plan. Actively promote the best collaboration in the team and take the initiative to address problems that may arise. Ensure that the requested project outcomes are delivered in due time.”
Ricardo Cruz	Product Owner	“Represents the interests of the stakeholders. Has a deep understand of the product and the application domain; the team will turn into the Product Owner to clarify the questions about expected product features. Should be involved in accepting the solution increments.”
Pedro	QA	“Responsible, in articulation with other roles, to promote the quality

Amaral	Engineer	assurance practices and put in practice instruments to measure que quality of the deployment.”
Rafael Baptista	DevOps Master	“Responsible for the development and production infrastructure and required configurations. Ensures that the development framework works properly. Leads the preparing the deployment machine(s)/containers, git repository, cloud infrastructure, databases operations, etc.”
All	Developer	“ALL members contribute to the development tasks.”

1.2 Agile backlog management and work assignment

To manage our backlog, we will make use of the tool Jira. We will have our backlog divided into 4 sprints of one week each given the timeframe of the project. In each sprint there will be a set of user stories, tasks and bug fixes that will be selected to be implemented. These items will initially be on the “TO DO” column and when a developer starts working on a given item it should be moved into the “IN PROGRESS” column. When the item is finished it should be passed into the “DONE” section. Besides the title and the description each item will have an assignee, zero or more co-assignees, one or more labels and a story point estimate from 1 to 4 depending on the effort required to perform the task, 1 being the easiest and 4 the hardest.
https://pedromiglou.atlassian.net/jira/software/projects/TQSPROJ/boards/4

2 Code quality management

2.1 Guidelines for contributors (coding style)

In this project we decided to use the Mozilla coding style available in the following link: https://firefox-source-docs.mozilla.org/code-quality/coding-style/index.html.

2.2 Code quality metrics

[Description of practices defined in the project for *static code analysis* and associated resources.]
[Which quality gates were defined? What was the r[ationale?]

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?]
We will use the gitflow worflow:

- Master – stores the official release history.
- Develop – stores the in-development version of the product.
- Release – stores a checkpoint of the develop branch and until it is merged to master it should only receive documentation and bug fixes
- FeatureX – branch where a developer develops a certain feature.

[Description of the practices defined in the project for *code review* and associated resources.]
We will use merge requests to review code, the changes must be approved by at least one other member before being merged to another branch.

[What is your team "Definition of done" for a user story?]
A User Story will be considered done if it meets all the acceptance criteria, has unit tests and integration tests written, executed, and passed, and is successfully deployed in a production environment.

3.2 CI/CD pipeline and tools

To implement a CI pipeline, Gitlab CI was used. The pipeline contains 2 stages, in the first the spring boot tests are run and a static analysis of the code with "sonarcloud" is done both for the spring boot projects and for the react projects. Here the pipeline fails if one of the tests fails or one of the projects breaks its quality gate. Then both web applications will be built for production breaking the pipeline if an error happens or a warning appears.
[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

To develop each API, a TDD (Test-driven development) strategy was adopted. First, a simple structure of the API was coded. Then for each new feature first the needed tests are implemented followed by the feature so that it satisfies all acceptance criteria. The tests are implemented using JUnit5, Mockito when we need to simulate dependencies and Spring MVC when we need to implemented controller tests.
To develop each web application, a BDD (Behavior-driven Development) strategy was adopted. First Cucumber was used to define the user stories that were needed. Then the feature in the webpage is developed having a user story in mind. Finally, that user story is recorded with Selenium IDE and adapted to support Cucumber.
[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?...]

[it is not to write here the contents of the tests, but to explain the policies/practices adopted and generate evidence that the test results are being considered in the IC process.]

4.2 Functional testing/acceptance

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

4.3 Unit tests

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

4.4 System and integration testing

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

4.5 Performance testing [Optional]

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