universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

João Farias [98679], Diogo Monteiro [97606], Marta Fradique[98629], André Freixo[98495]

Final Version

1 Project management

1.1 Team and roles

To improve our group work dynamic, each member of the group has different responsibilities and positions.

- Team Coordinator: João Farias
- Product Owner: André Freixo
- QA Engineer: Marta Fradique
- DevOps master: Diogo Monteiro

1.2 Agile backlog management and work assignment

To handle the backlog management, we used Jira, this platform allowed us to create issues, which were the user stories of our application. Then we would create weekly sprints with our tasks that must be developed in the one week time period. In the beginning of the week we had a meeting to distribute the tasks equally and to discuss any issues of the previous week. In case a member of the group ran off tasks during the week, new ones were added to the sprint.

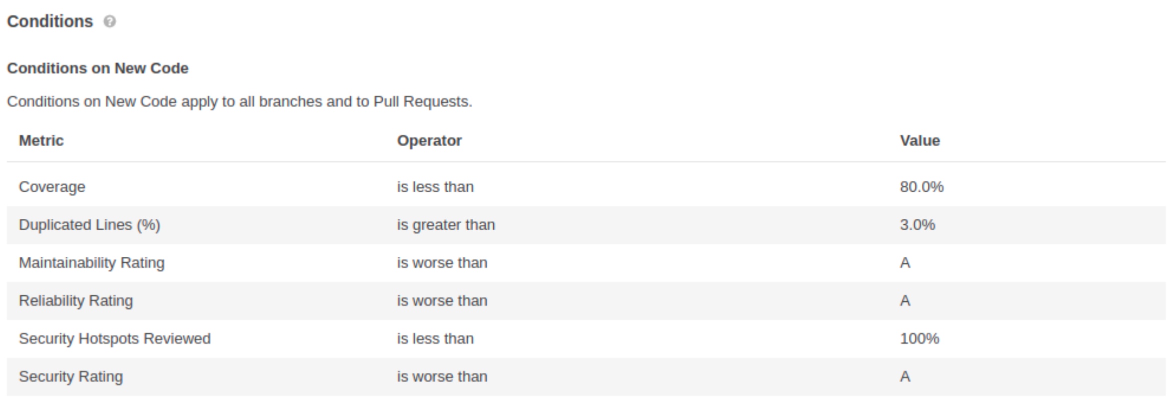
2 Code quality management

2.1 Guidelines for contributors (coding style)

As we were developing this project the group members adopted some practices to have a good quality code. Some of them were to comment on the code, use a correct indentation, document the API to know what each endpoint does, use intuitive names for variables, use SonarCloud to detect errors, etc.

2.2 Code quality metrics

To perform the statics analysis of the code, we used SonarCloud, this tool is very useful since it performs automatic revisions of the code in order to find bugs, code smells and other metrics specified in the following figure. We configure SonarCloud with the default values, once it is the most



Conditions

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Fig 1- SonarCloud - Default Quality Gates

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

To develop our work continuously and with quality we adopted some practices such as each time a member started to develop a new feature(Issue created in Jira), he created a new branch and when the task was complete he would perform a pull request in github, posteriorly, another member of the group would review the code and accept it in case everything was working and developed properly.
Each group member would consider the task as completed when the features were implemented, working and the respective tests were performed and passed successfully.

3.1 CI/CD pipeline and tools

To build our program with a continuous integration and increments we used GitHub Actions. Using this tool each time a member did a push command or a pull request to the main brach, the *SonarCloud* mechanism performed an automatic static revision of the code, with the statists feedback of SonarCloud we were able to upgrade our code, eliminating bugs and vulnerabilities.

4 Software testing

4.1 Overall strategy for testing

As we were developing this project we used the methodology TDD(Test-Driven Development), with this strategy we started by defining the user stories, then we wrote the tests, and only after these two phases are completed, the code for the task is written. When we were starting this project we used this methodology, but then since we were overtaken by the work of this course and the others, we started using less of this practice. Now when all the work is finished we regret not using this technique until the end, because it really upgraded the level of the written code.

4.2 Functional testing/acceptance

Due to the lack of time we weren't able to perform neither Selenium tests or Cucumber tests, which really disappointed us since that was one of our initial goals.

4.3 Unit tests

In this project we used *Unit Tests* to test an isolated part of the system, the goal is to ensure that the basic components of the system work correctly and as expected.

The components of the system where we decided to perform this type of tests were in the *Service*, *Controllers*, *Repository* and *Models.*

In the *Service*, to test the components we used the extension *Mockito*, we did that in order to be able test the methods independently of the persistence lawyer, therefore, we used *mocks* instead of the *repository*. As we were doing the tests we tried to make tests of multiple scenarios especially where error may occur.

In order to test the repositories we used @DataJpaTest with the database TestContainers, and tried to develop as many tests as possible, always aiming especially for the scenarios where errors may occur.

On the Controllers the Unit Tests were developed using MockMvc, this allowed us to use mocks instead of the Services, isolating the Controllers behavior, and granted us a better testing environment. Then, as we did with the other components we tested different scenarios specially where error may occur.

On the models we added data to the repository and verified if the data saved in the database matched the inserted data.

4.4 System and integration testing

To confirm that all the different components that were previously tested individually, had the correct behavior together in the application we performed new tests. In these tests it wasn't used *Mock* to substitute the persistence lawyer, and the true persistence developed by us was used.
These tests were performed in order to test the Controller endpoints, to check if the behavior with the Rest API was correct.

References

[1] *SonarCloud* definition

https://sonarcloud.io/?gads_campaign=Europe-3-SonarClouds&gads_ad_group=SonarCloud&gads_keyword=sonarcloud&gclid=Cj0KCQjwntCVBhDdARIsAMEwACINwPPK5Yq5aE2zsnAW96gbmeuQFUvNX6o29uduoosNlqzdUwl31KQaAjyGEALw_wcB

[1] *TDD definition*

https://www.guru99.com/test-driven-development.html