TQS: Quality Assurance manual

Diogo Cunha [95278], Pedro Tavares [93103]
Gonçalo Pereira [93310], José Carlos [87456], , Juan Lessa [91359]
v2021-06-21

1 Project management

1.1 Team and roles

Pedro is the Team Leader, who has the responsibility of distributing the work between the team, dealing also with the project calendar, relative to the outcomes that must be delivered in each iteration.

Gonçalo is the Product Owner, giving orientations to the team and responsible for representing the stakeholders.

Juan is the DevOps Master, his role was to ensure the deployment of the product, as well of the git repository, configuring the development and production infrastructure.
Diogo and José are responsible for the Quality Assurance of the product.
All members contribute to the development process of the system.

1.2 Agile backlog management and work assignment

Our backlog is user stories oriented and those user stories are being created in github with the github issues. We have labels to define user stories from tasks, we can assign a card to a person in the repository, and a project board for every iteration. The project board has a to-do list, in progress, review in progress, reviewer approved and done and in those boards we can add user stories and tasks.
Although only two members only tried to use it, we tried to have a board for every iteration.

2 Code quality management

2.1 Guidelines for contributors (coding style)

We didn't decide on a coding style, we just tried to make the backend have similar styles by reviewing code on pull request and then refactoring the code together, this in the initial part of the project.

2.2 Code quality metrics

For the static code analysis two different approaches were idealized: the first one, considering our a local solution, where the IDE tools were considered, using the code inspection available in Intellij IDE); the second considered was Sonarcloud, used as a behind check, that is, before the code was committed or merged, a manual verification with Sonarcloud was done. After the quality gates were graded, the new code could be pushed. When that new code was merged, the Continuous Integration methods automatically contacted the sonarcloud to verify again that code against the quality gates..

The initial quality gate was:

Blocker Issues	is greater than	0
Coverage	is less than	80.0%
Critical Issues	is greater than	0
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Major Issues	is greater than	0
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

We chose to adopt GitHub flow as our tool of keeping track of Development workflow.

GitHub has a lot of tools that help the Continuous delivery pipeline, for example the Issues, that can be very helpful. An Issue can be a user story, feature, etc, and it is something that can be used to keep track of what is most important to implement right now.

To assure maximum quality in our project, we made sure that to commit to the develop branch each change should be reviewed and approved by another member of the group, using the pull request restrictions GitHub offers.

3.2 CI/CD pipeline and tools

The CI pipeline was to verify the increments and the features added, whether that be through pull requests or pushes, where the tests were run and the build was verified. Whether that be integration or unit tests, all the tests must pass. Sonarcloud was also considered to verify the quality gates of the new code.

The CD pipeline is responsible to be run by the VM, generating in product a new artifact, using a docker container to deploy it.

4 Software testing

4.1 Overall strategy for testing

The testing strategies for the project was TDD.
We implemented the controllers and then tested them by making valid and invalid requests and by expecting exceptions and different http status codes. We tried to test all the code and all the conditions that could impact the output.
We only used mock mvc to test.

4.2 Functional testing/acceptance

We unfortunately didn't make any functional tests.

4.3 Unit tests

Our unit tests were abundant and were made during the development. We tried to test every condition in the code.
We have unit tests for all controllers and services associated. For every endpoint and service function we have at least one test with a valid input and a test that throws an exception or that makes the output different.

4.4 System and integration testing

Regarding the integration test we only have those kinds of tests for the Business API. We instantiate a new docker container with mysql only for test purposes.

Those tests are divided in two classes, one to test the address endpoints, and the other where we use @TestMethodOrder and we simulate the flow of the application, where the user registers, logs in, and adds some products to the cart.