TQS: Quality Assurance Manual

Gonçalo Passos [88864], João Vasconcelos [89022], Luís Valentim [93989], Marta Ferreira [88830], Rita Ferrolho [88822]

v2021-06-08

1 Project Management

1.1 Team and Roles

Team Coordinator	Marta Ferreira
Product Owner	Luís Valentim
DevOps Master	Gonçalo Passos
QA Engineer	João Vasconcelos, Rita Ferrolho

1.2 Agile backlog management and work assignment

For the backlog management, GitHub's kanban tool is being used in order for everyone to be in sync with what work has to be done for a certain sprint, what issues have already started development and who's working on each one and what issues have been closed and pull request linkage.

2 Code Quality Management

2.1 Guidelines for contributors

We adopted standard Java code guidelines in order to have easily readable code, which follows the naming and organization procedures found within this manual used by Google.
https://google.github.io/styleguide/javaguide.html

2.2 Code quality metrics

Metrics which judge the project's quality, accessed by SonarQube on the process of static code analysis, some of the more relevant ones we intend to use as our quality gates for quality assurance are:

Code Coverage - The percentage of code covered by automatic testes
Nº Code Smells - How many code smells are in the project
Nº Buggs - How many bugs are in the project.
Nº Vulnerabilities - How many vulnerabilities are in the project.
Duplication - Percentage of duplicated code.
Technical Debt - Average amount of time needed to fix the current problems within a project.

3 Continuous Delivery Pipeline (CI/CD)

3.1 Development workflow

For workflow development the TDD (test-driven development) strategy will be used. Basically, tests will be developed and run before implementing final versions of the code that allows them to pass the tests. The tests guide the development instead of being an afterthought.

Each pull request will be submitted to a pipeline which automatically runs tests, runs static code analysis and retrieves to the developers the problems found within the solution, this automation process is very important in quality assurance due to the agility it provides to the process as well as keeping always available whether or not the current build is functional and if it isn't where is it failing which is very useful for the QA engineers.

3.2 CI/CD pipeline and tools

With the help of GitHub actions, a workflow with commands will be defined to, test, analyze and deploy the project. This is done with automatic tests for each commit, as well as run static code analysis for the project and assure the deployment doesn't have any unexpected errors, in a controlled environment which evaluates the given results. It is necessary to keep a mentality where development and peer review occur frequently.

4 Software Testing

4.1 Overall strategy for testing

Always follow the approach Arrange, Act and Assert. Prepare the elements of the situation that will be simulated, execute the action to be tested, verify if the results occurred as intended. Don't mix different types of tests.

4.2 Functional testing/acceptance

Selenium will be used to verify that the system accomplishes the requirements previously defined.

We're following a whitebox approach since these are meant to verify everything is working fine technically and does not require user inexperience to be taken into account

4.3 Unit tests

Test each component to verify that the code follows the expected procedure, with exception of the classes that save the information. It is not relevant to test getters and setters.

We're following a whitebox approach since these are meant to verify everything is working fine technically and does not require user inexperience to be taken into account

4.4 System and integration testing

Verify that all components interact with each other as expected, by using mocks. Verify that the system, when deployed as a whole, behaves in a predictable manner, which can be evaluated by using Selenium.

We're following a whitebox approach since these are meant to verify everything is working fine technically and does not require user inexperience to be taken into account

4.5 Usability Tests

Usability tests, although not automatic as their peers, are very relevant for quality assurance as they evaluate how well a user can use the system, how easy it is to learn and overall if it's ready to be deployed in a real consumer market.

These should have as diverse of a user base as possible in order to properly verify the system's usability and follow a blackbox approach with no extra information in order to assess how easily users can understand what's being presented and learn how to use the system.