deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

Alexandre Lopes [88969], André Amarante [89198], João Nogueira [89262], Tiago Melo [89005]
v2020-05-21

1 Project management

1.1 Team and roles

In order to have a more organized team and maintain the focus of each member in their tasks, we decided to assign each one of us a specific role. This way, each team member can focus on their subset of tasks, which are related to his role, instead of getting through a frequent context switch, which is very inefficient. So, we assigned four different roles: team manager, product owner, DevOps master, and developer.

André was assigned to the team manager and, as so, he is responsible to manage the team workload, ensure the priorities are well defined and ensure their concretization in time, as well as promote good collaboration between team members in order to have the expected project outcomes.

Alexandre is the product owner, as he is who came up with the idea and concept for the application. As the product owner, Alexandre will be involved in accepting the solution increments and make clear to the group what are the expected product features and how they shall behave.

João and Tiago were both assigned to the DevOps master role as this one is more complex and critical to the whole functioning of the application. Tiago and João will be in charge of setting up the infrastructure and its configurations. This includes configuring the CI/CD pipeline, placing the git repository, preparing the deployment environment (containers, VMs, etc) and more.

Finally, all four members were also developers, contributing to each development task in order to meet the user stories and the project's requirements. Alexandre and André focused more on the frontend of the application (developing UI, assuring a good user experience, displaying the correct data to the user that came from the backend, etc), while João and Tiago contributed more to the backend (development of the API, database configuration, etc). Also, Alexandre and André were who developed the mobile app for the external client module, which makes use of the developed API.

1.2 Agile backlog management and work assignment

As the tasks required to accomplish the goals and requirements for this project are so many and can be diverse between team members, in order to keep track of what is done, what is being done and what needs to be addressed next, we decided to use Trello with Agile Tools. With that, we were able to raise new tasks, assign to each team member, and establish deadlines in some tasks, according to the iteration plan defined for the project. Besides the internal tasks, we also used Agile Tools add-on on Trello to help us to define, prioritize and estimate user stories based on the project schedule and on the effort needed to have the functionalities that meet each user story. In each iteration, we looked at the backlog and addressed the user stories with most priority

Besides Treelo, we also used Slack to make the team synchronized and aware of what was being done at each time, as well as schedule regular meetings in order to each team member present to the rest of the team what has been working on and, as a team, define new goals and which steps that should be taken next.

The project outcomes were also shared between team members through Google Drive (such as meeting outcomes, project reports, etc) and through the git repository.

2 Code quality management

2.1 Guidelines for contributors (coding style)

During the development of this project, which was developed in Java, we followed the main naming conventions of this language, such as class names with first letters in uppercase, public fields named

with lowercase, constants with all caps, and underscores, etc. We also followed some generic programming standards and good practices, such as handling possible errors through catching exceptions and dealing with null values, providing useful error messages, avoiding redundant or very extensive code that can be inefficient, writing comments, adding TODOs to the code, etc. These practices are in line with the Java Coding Style Rules of the AOS project and the main language standards, and the team was guided and inspired by them and didn't define any major new internal guidelines.

Besides those practices, the team was also encouraged to adopt the same software design patterns as needed. The rationale of this approach is to have a more consistent and maintainable code that can be easily read, easily corrected and also less error-prone.

2.2 Code quality metrics

In order to have a reliable measurement of the quality of the developed code, we took advantage of static code analysis tools, mainly SonarQube and JaCoCo plugin. These tools allowed us to see how robust our code was in terms of security, code smells/good practices, testing coverage and so on. To do that, we defined the following quality gates, that defined the conditions under which our code would pass during the analysis:

- Code coverage shall be greater than 50%
- Duplicated lines shall be less than 3%
- Maintainability rating shall be A
- Reliability rating shall be A
- Security rating shall be A

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

The chosen workflow to be applied during the project's development was the GitHub Flow.
It focuses on feature-branching, pull-requests, code-review and continuous deployment and merging. Essentially, a new branch is created for each new feature to be developed. These branches help maintain the master version stable and clean, allowing experimentation and providing modularity. Each branch will have a series of commits with concise, meaningful commit messages. This allows us to keep track of the changes being made and rollback to a previous point if necessary. After the branch has been worked on, a merge/pull request (GitLab vs GitHub terminology) is opened and the code is reviewed among peers. Here we can keep track of the current changes, commenting the code or asking for feedback. In this project the code review is done in pairs, meaning that each member of the backend and frontend teams reviews each other's code. For a merge request to be accepted the following criteria are evaluated:

- Coding standards
- Clear commit messages
- Code organization and repetition
- Unit tests and BDD (developing and committing those tests, before the actual implementation).
- SonarCloud issues.

All merge requests are also notified in the team's Slack Workspace, with the proper gitlab integration configurations.

Once a merge request is accepted, we are positive the new code is tested and working, and it is then merged with the project's master branch.

Each feature-branch consists of processed user stories. It works as follows:

1. A user story is discussed with the Product Owner and within the development team.
2. The user story is created, stored in the project's backlog (Trello) and given a priority and points based on the difficulty of the given task.
3. The story is decomposed into features which are placed in the 'To Do' card and assigned to the developers.
4. Features are developed in their own branches, following the aforementioned GitHub Flow.
5. A merge request for the feature-branch results in code reviewing according to the above standards and requirements.

Once the corresponding feature-branch has been merged with the master version the user story is considered to be finished, given it successfully passed the previously referred peer-review, acceptance criteria and unit testing.

3.2 CI/CD pipeline and tools

In order to promote and have easier code reviews, developers should stick to making short, yet frequent commits, rather than long sporadic ones. This eases the load on the code reviewer side and allows developers to ensure the codebase is kept up with the established standards. For the code to be acceptable into the codebase, it needs to be tested which, ideally, is done automatically. One way to achieve this is by implementing a Continuous Integration pipeline, which asserts whether or not a given build passes or fails whenever a commit is pushed.

The tool used for the CI pipeline was the GitLab CI system. By providing Runners and containers where the code will run and be tested, we can configure a CI pipeline file to set up the stages and phases of our pipeline. In case one of them fails, the rest of the pipeline isn't executed and developers can check the logs and troubleshoot.

In an earlier stage, our pipeline consisted of three very basic stages, concerning only the maven project itself:

- Build: the code was compiled using $mvn clean compile
- Test: which maps to the maven goal of $mvn test
- Deploy: which consists of packaging the application

Currently, the CI pipeline has these four stages:

- Connect: An image of the chosen project's database, mysql, is initialized with the proper configuration settings, creating the database and mysql-users. We can now execute the remaining steps while maintaining database connection.
- Build: executes the maven goal $mvn clean compile, compiling all the project's source code.
- Test: which runs $mvn test, therefore running all our designed Unit and Integration tests.
- Package: running $mvn package and generating the project's .jar file.

SonarCloud integration is also configured in GitLab's CI file, so every branch is checked whenever any new code is committed.

Furthermore, any broken (failed) pipelines are notified on the team's Slack Workspace.

The deployment is made with standard docker containers, running the application (Spring Boot project + MySQL database) on a Virtual Machine provided by the course instructor.

4 Software testing

1.1 Overall strategy for testing

For the tests, the general strategy used was Test Driven Development, with a general set of each type of tests developed before the actual functionalities were put in place. This allows the team to first specify in test form how a functionality should behave and then implement it, knowing that it will be finished once the test passes. The team also opted to use Behavior Driven Development with Cucumber to validate the core user stories, which was another way to create higher level functionality testing before the actual development. All four team members were involved in the continuous testing phase of the project.

1. Functional testing/acceptance

Functional tests were developed using the Selenium Webdriver tool, which allowed team members to record user interface interactions, add manual increments to the tests and export them to the projects testing language (Java). These tests were developed once the project user interface was finished. The executed plan was to have the team focused on covering as much of the user interaction points as possible, focusing first on the main user interactions flows, followed after by secondary ones.

2. Unit tests

Unit testing was developed using the JUnit framework which all team members were already familiar with. These tests were created from a developer perspective. Therefore, it was expected and achieved a coverage of all developer-created methods leaving things like getters, setters and constructors untested as they already obey a tested schema.

3. System and integration testing

API testing was developed using Spring MockMvc and REST Client (TestRestTemplate).

4. Performance testing

If performance tests are written, they will be created using the JMeter tool