TQS : Quality manual and continuous delivery

Project: TechFlux

Group:

- Aneta Pawelec
- Gonçalo Almeida
- Pedro Candoso
- Ricardo Antão

2- Project Management

Team and roles

Aneta Pawelec - DevOps Master
Gonçalo Almeida - Team Manager
Pedro Candoso - Product Owner
Ricardo Antão

In our project, we all assume the role of developers and reviewers because, given the size and scope of the application, we all will contribute in the writing of code as well as reviewing the code produced by the other members.

Backlog management and work assignments

Backlog management is accomplished with the usage of PivotalTracker and implements the following workflow:

1. Write stories - Any member of the development team can add new story features to the project

2. Discussion - At this point the team will discuss the various stories to share their understanding and to prioritize each one

3. Start stories - A developer or group of developers will start working on a story that is ready.

4. Finish and deliver stories - As soon as the story is complete and tested, it should be marked as finished by the developer.

5. Accept or reject stories - A member of the development team will verify whether acceptance criteria has been met and accept or reject the story.

Reference: https://www.pivotaltracker.com/help/articles/workflow_overview/

Issues tracking system

PivotalTracker will be the tool chosen to keep the list of issues with the project. The workflow for solving issues will be similar to the one used for backlog management. A developer that finds a problem should add it to the backlog and give it a priority, marking it as an issue. In case this issue is associated with one or more stories, a link between them should be made in the description of the issue presented. One of the developers will then start working on a fix and mark it as finished once proper testing has been successfully completed.

3- Code management and quality

Contributor guide (*coding style)*

Java code style based on Android Open Source Project (AOSP) following rules such as:

- Don’t ignore exceptions
- Don’t catch generic exceptions
- Don’t use finalizers
- Fully qualify imports
- Use Javadoc standard comments
- Write short methods
- Define fields in standard places
- Limit variable scope
- Order import statements
- Follow field naming conventions
- Use standard brace style
- Limit line length
- Use standard Java annotations
- Treat acronyms as words
- Use TODO comments
- Use logs sparingly
- Be consistent

Reference: https://source.android.com/setup/contribute/code-style

SCM *workflow using GitHub Actions*

Git feature branching workflow :

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch. This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the master branch will never contain broken code, which is a huge advantage for continuous integration environments.

The Git Feature Branch Workflow is a composable workflow that can be leveraged by other high-level Git workflows.. Git Feature Branch Workflow is branching model focused, meaning that it is a guiding framework for managing and creating branches.

The Feature Branch Workflow assumes a central repository, and master represents the official project history. Instead of committing directly on their local master branch, we

will create a new branch every time we start work on a new feature. Feature branches should have descriptive names, like user-menu-items or issue-#1061. The idea is to give a clear, highly-focused purpose to each branch. Git makes no technical distinction between the master branch and feature branches, so we can edit, stage, and commit changes to a feature branch.

In addition, feature branches can (and should) be pushed to the central repository. This makes it possible to share a feature among us without touching any official code. Since master is the only "special" branch, storing several feature branches on the central repository doesn't pose any problems. Of course, this is also a convenient way to back up everybody's local commits.

Code Review

GitHub - everyone of team members are developers as well as code reviewers. Checking code of colleagues allows us not only to optimize and increase readability of code but also increase our overall insight into the project. What is more, everyone can learn new things from other branches that they are working on. In our case it might happen that we will need to integrate some parts of code across 2 different fields managed by different team members. In that case explaining to each other all code and reviewing it together will be a benefit.

Static Analysis

We used SonarCloud (with configurable quality gates usage) for static analize as well as Code inspection in IntelliJ, intending to correct and clean the produced code, attending to bad smells or bugs that might be detected by the platform in an autonomous integration.

We plan to follow some of the industry standards when producing code, like factoring and limit the size of functions (with some exceptions if needed).

"Static analysis tools compare favorably to manual reviews because they're faster, which means they can evaluate programs much more frequently, and they encapsulate some of the knowledge required to perform this type of code analysis in a way that it isn't require the tool operator to have the same level of expertise as a human auditor. " - An overview on the Static Code Analysis approach in Software Development, FE

4- Continuous integration & continuous delivery

Continuous integration practice
1. Each of us commit to a shared repository regularly
2. Changes in SCM are observed and trigger automatically builds
3. Immediate feedback on build failure (broken builds have high-priority)
4. Sharing repositories.

Based on Fowler's CI practices that we would like to implement :
1.Maintain a Single Source Repository
2. Automate the Build
3. Make Your Build Self-Testing
4. Keep the Build Fast
5. Test in a Clone of the Production Environment
6. Make it Easy for Anyone to Get the Latest Executable
7.. Everyone can see what's happening
8. Automate Deployment

With containers, we intend to have a lightweight service that allows the use of different configurations and scenarios for the tests proposed. Some of the scenarios are testing the environment in different OS, browsers, test the API with different resources and, if possible, Gmake some performance tests based on concurrence or user access.

5- Tests

This section explains different test techniques, with which tools they will be implemented and gives a small description of how to proceed and use them.

Functional testing

Setup: Selenium + Junit

With this setup, we will test the front end of the application. Selenium allows us to test the relations between pages, checking their content and validating if the information retrieved is correct.

Using this setup, we intend to follow a black box approach in the generation of the tests, meaning that we do not need access to the source code to test the application, this allows other users to test the application and report bugs if without the source code and with their own devices.

This approach is recommended when doing functional tests because the review of the application can be done by outside people, while sometimes the developers might introduce some incorrect behavior without noticing it, this behavior can be easily detected by "other pair of eyes".

"The main focus in black box testing is on the functionality of the system as a whole. The term **'behavioral testing'** is also used for black box testing. Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged." - Black box testing: An in-dept tutorial with examples and techniques

- Examination of requirements and specifications of the system
- Testers use some valid and invalid inputs
- Determination expected outputs for all those inputs
- Constructing test cases with the selected inputs
- Executing test cases
- Comparing the actual outputs with the expected outputs
- Report defects
- Re-test

Unit testing

Setup: JUnit

With this setup, we will test the functionality of the app which means the operations like posting a flat rent offer, search results for a specific city, this allow us to control the how the app should work. We decided to use this setup because the website will be created with

Angular, and the most recent documentation of Angular recommends the usage of this tools.

With a white box approach, the tester has access to the source code and should make operation focussed tests, such as, test the results of a query or the components of the application.

For openbox using Junit with practices like Statement Coverage and Branch Coverage.

Code is verified according to documented specifications, testing focuses on boosting security, improving the flow of inputs and outputs and on improving usability and design. It uncovers application vulnerabilities.

System and integration testing

Setup: Spring Boot Testing

With this setup, we can test the API endpoints and to understand if the data requested is the expected.

We can make the tests using both white or black box scenarios, which means that the tests applied to the API can be done without knowledge of how the functions work, but it can also be done with mocking techniques, this way, we can create faster tests without the need of a database.

This approach is beneficial to the developers in the sense that we can work with data before setting up a database, even though it does not give a proper test, it simulates how the application should work.