TQS: Manual de Garantia de Qualidade

Conteúdo

[**TQS: Manual de Garantia de Qualidade 1**](#_heading=h.gjdgxs)

[**1**](#_heading=h.30j0zll) **Gestão do projeto 1**

[1.1](#_heading=h.1fob9te) Equipa e papéis 1

[1.2](#_heading=h.3znysh7) Gestão do backlog Agile 1

[**2**](#_heading=h.2et92p0) **Gestão da qualidade de código 2**

[2.1](#_heading=h.tyjcwt) Estilo de código 2

[2.2](#_heading=h.3dy6vkm) Medidas da qualidade de código 2

[***3***](#_heading=h.1t3h5sf) ***Continuous delivery pipeline (CI/CD)* 2**

[3.1](#_heading=h.4d34og8) *Workflow* de desenvolvimento 2

[*3.2*](#_heading=h.17dp8vu) *CI/CD pipeline* 3

[3.3](#_heading=h.3rdcrjn) Artifacts repository [Optional] 3

[**4**](#_heading=h.26in1rg) **Testes de Software 3**

4[.1](#_heading=h.lnxbz9) Estratégia para os testes 3

[1.](#_heading=h.35nkun2) Testes funcionais 3

[2.](#_heading=h.1ksv4uv) Testes unitários 3

[3.](#_heading=h.44sinio) Testes de sistema e integração 4

[4.](#_heading=h.2jxsxqh) Testes de performance 4

# Gestão do projeto

## Equipa e papéis

A nossa equipa é constituída por quatro membros:

Dono do produto: [João Carvalho](https://github.com/joaocarvalho19) - 89059

*DevOps*: [Vinícius Ribeiro](https://github.com/viniciusbenite) - 82773

*DevOps*: Bernardo Rodrigues - 88835

Gestor da equipa: [Alina Yanchuk](https://github.com/alina-yanchuk02) – 89093

Sendo que, também somos todos desenvolvedores.

## Gestão do backlog Agile

Para gerir o backlog do projeto, adotamos uma prática baseada em *user stories*, sendo que estas são descritas e colocadas, por ordem de prioridade, no Pivotal Tracker.

Link para o Pivotal Tracker do projeto: <https://www.pivotaltracker.com/projects/2448629>

Iremos implementar as *user stories* com maior prioridade primeiro, seguindo uma metodologia Agile, de planeamento, desenho, implementação, testes e deploy recorrentes.

# Gestão da qualidade de código

## Estilo de código

Iremos adotar um estilo de código universal para todos os contribuidores do projeto, de modo a garantir a qualidade de leitura e compreensão do código.

É privilegiada a escrita de comentários, para explicar o que foi feito, a utilização de nomes para variáveis e funções em Inglês e a criação de ficheiros separados para cada componente, de modo a manter tudo organizado e facilmente acessível e encontrado.

Como guidelines para a escrita de Código, iremos usar as normas recomendadas pela Google:

<https://google.github.io/styleguide/javaguide.html>

<https://google.github.io/styleguide/htmlcssguide.html>

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

## Medidas da qualidade de código

Para analisar estaticamente a escrita do código, optamos pelo uso do SonarQube, recusando o uso de qualquer acréscimo de código que obtenha uma classificação de qualidade pouco favorável.

Através do SonarQube, iremos conseguir visualizar em que estado se encontra a qualidade do nosso projeto, procurando sempre melhorá-lo.

Para esse projeto, nós adotamos um coverage mínimo de 25%.

# *Continuous delivery pipeline (CI/CD)*

## *Workflow* de desenvolvimento

Durante o workflow do nosso projeto, tendo sido adotado o GitHub flow, vamos proceder à criação de várias branches, com a intenção de cada um ter, como objetivo, a criação de uma feature distinta; esta feature vai ter uma ou mais user stories ao qual é relevante,

Quando a feature da branch for concretizada, prosseguiremos para a realização de um pull request que passará por um processo de análise/discussão e code review entre pares, se tal for pedido; após aprovação , será então deployed e, após passar nos testes, será realizado o merge.

<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

<https://guides.github.com/introduction/flow/>

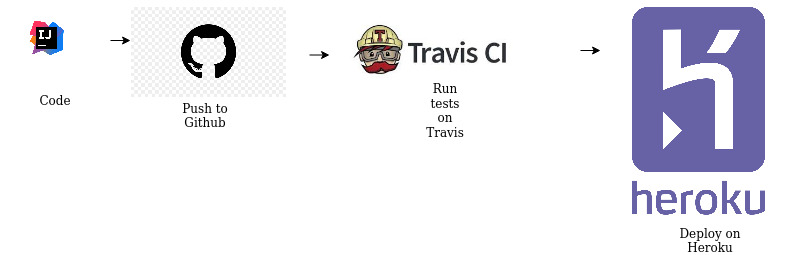
Haverá uma lista de critérios que serão necessários para declarar uma *user story* com realizada, nomeadamente os critérios de aceitação cumpridos, o dono do produto aprovou-a, testes realizados e passados.

Temos 3 branches principais:

* Features branch: aqui é concentrado todo o desenvolvimento da aplicação. Para cada nova feature é criada uma nova branche. Após passar nos testes de CI, e outro membro do time fazer o code review, fazemos merge com a branch features, para depois, fazer o merge na master;
* Documents branch: aqui concentramos toda a documentação relevante ao projeto;
* Bug fixes: qualquer alteração de código que não seja uma nova feature é colocada aqui.

## *CI/CD pipeline*

Para o controle do Continuous Integration, nós usamos a ferramenta Travis CI. Cada novo push de código feito ao git despoleta uma checagem pelo Travis, que por sua vez, executa todos os testes feitos por nós e cria um novo .jar para a aplicação. Em caso de sucesso, o Travis fará o deploy da aplicação no Heroku, de forma automática.



O deploy pode ser consultado em: tqs-final-project-barbershop.herokuapp.com/greeting

# Testes de Software

## 4.1 Estratégia para os testes

Irão ser escritos testes unitários, de modo a testar todas as classes e camadas da aplicação, e termos uma percentagem de *coverage* bastante alta; testes funcionais para testar as nossas páginas Web; e teste de integração para testar todas as *API’s* feitas.

Para testar os controllers, repositórios e a REST API, usamos teste unitários, com auxílio das ferramentas Junit e Mockito. Para o code coverage usamos JaCoCo e SonarCloud. Além disso, usaremos Selenium Webdrive para os testes da interface do usuário.

## Testes funcionais

[Project policy for writing functional tests (closed box, user perspective) and associated resources.]

## Testes unitários

[Project policy for writing unit tests (open box, developer perspective) and associated resources.]

## Testes de sistema e integração

[Project policy for writing integration tests (open or closed box, developer perspective) and associated resources.]

API testing

## Testes de performance

[Project policy for writing performance tests and associated resources.]