# TQS: Manual de Qualidade

***Dinis Cruz [93080], Duarte Mortágua [92963], José Sousa [93019], Tiago Oliveira [93456]***
v2021-06-21



# 1 Gerenciação do projeto

## 1.1 Equipa e designações

Equipa:

- Team Lider: Duarte Mortágua, assegurar que há uma boa distribuição das tarefas e que estas estão a decorrer como planeado. Promover uma boa colaboração entre a equipa e ter iniciativa para resolver problemas que possam existir. Assegurar que o trabalho é entregue quando necessário.
- Product owner: Tiago Oliveira, representa os interesses do cliente. Tem um conhecimento extenso daquilo que é pretendido do produto, a equipa recorrerá a este membro quando houver dúvidas sobre funcionalidades da aplicação. Deve estar presente na entrega de incrementos
- QA Engineer: Dinis Cruz, responsável de promoção quanto as praticas para assegurar bom software desenvolvido.
- DevOps master: José Sousa, responsável pela estrutura de desenvolvimento e de produção. Prepara os ambientes para serem deployed.
- Developer: Todos os membros vão ser desenvolvedores da plataforma.

## 1.2 Gerenciamento do backlog e distribuição do trabalho

Criar as user stories mediante aquilo que será pretendido da nossa aplicação e usar a Zen Board para estarmos cientes daquilo que já foi feito e aquilo que ainda não está completo.

Também faz sentido criar um link entre a Zen Board e aquilo que se passa no repositório do git hub para termos a certeza de que há coerência entre as 2 ferramentas que estamos a usar
O work flow normal não deverá fugir muito de:

**Criar User Story**

1. Entrar na Board do ZenHub.
2. Canto superior direito -> New Issue.
3. Template -> Story.
4. Preencher título da story, descrição, acceptance criteria, definition of done.
5. Atribuir assignees, sprint e story points (estimate) no painel da direita.
6. Submit new issue.

**Desenvolver Story**

1. No repo local -> git checkout develop. git pull. (git flow init se nao tiverem feito). git flow feature start <#_user_story>, onde # é o número da user story na board do projeto. Se forem precisas mais features fazer feature <#_user_story_1>...
2. Desenvolver (add, commit, add commit...)... e ir preenchendo o acceptance criteria/definition of done no issue, no zenhub.

**Publicar Story**

1. No repo local -> git flow feature publish <#_user_story>.
2. No GitHub (não no ZenHub), fazer Compare & Pull Request da feature.
3. Mudar merge para o develop (e não para o master)
4. Comentar as mudanças.
5. Em baixo conectar o PR ao issue (aka user story) correspondente.
6. No painel da direita -> Team Workspace por ReviewQA ou Done (dependendo do caso), adicionar 2 reviewers.
7. Create pull request.
8. (Aqui vai entrar a CI pipeline)
9. (Pessoal aprova, da merge e delete do branch) e move a story para o Done na board do ZenHub.

# 2 Gerenciamento da qualidade do código

## 2.1 Guias para os colaboradores

### 3.2.1 Estrutura dos ficheiros:

No wildcard imports.
Exactly one top level class declaration.
Ordering of class contents.

### 3.2.1 Formatação:

Braces are used when optional.
One statement per line.
Enum classes line break.
Annotations on a line of its own.

#### 3.2.1 Nomes:

Class names in upper camel case.
Method name in lower camel case.
Constant names use CONSTANT_CASE: all uppercase letters, with each word separated from the next by a single underscore.

#### 3.2.1 Praticas de programação:

Caught exceptions not ignored.

### 2.2 Metricas de qualidade

Depois do sucesso que 3 dos 4 membros do grupo tiveram a usar sonar cloud decidimos usar de novo para analise estática de código.
Vamos usar a "quality gate" default visto que a nosso ver define boas praticas e para os nossos propósitos parece-nos que vamos conseguir criar software de qualidade com a ajuda desta métrica.

## 3 Continuous delivery pipeline (CI/CD)

### 3.1 Development workflow

O workflow completo já foi descrito acima.
Visto que somos um grupo pequeno achamos que 1 reviewer por "pull request" será o necessário para manter qualidade no código, esta decisão teve também em conta a confiança que já existe dentro do grupo depois de muitos projetos já desenvolvidos em conjunto. Isto pode ser facilmente alterado se alguém se juntar ao grupo e necessitarmos de alterar o nosso review process para mais pessoas.
A nossa definição de ter uma história feita é unit, integration e function testing feitos e revistos por um membro do grupo.

### 3.2 CI/CD pipeline e ferramentas

#### 3.2.2 CI Pipeline:

Feature workflow:

- Os branches de features (Spring) quando recebem um push correm um workflow que dá build e correm os testes da aplicação apresentando também resultados de analise estática de codigo.

Main workflow:

- Quando o develop e o master receber um merge correm a análise estática de código.

#### 3.2.2 CD Pipeline:

- O deploy está a ser feito no Heroku.
- Deploy é feito quando há merge para os branches develop de qualquer um dos
- Não é feito com GitHub actions, mas sim na plataforma do Heroku que permite apenas começar a compilar o código e a dar deploy apenas se a parte anterior de CI passar com sucesso.

# 4 Teste do sofware

## 4.1 Estrátégia geral para testagem

Inicialmente foi pensado o uso de TDD mas depois de uma má adesão do grupo ao mesmo optamos por fazer tradicional development onde primeiro implementamos o código e depois os testes, podendo posteriormente haver uma faze de refactoring de ambos. Foram misturadas diferentes ferramentas como REST-assured.

[it is not to write here the contents of the tests, but to explain the policies/practices adopted and generate evidence that the test results are being considered in the IC process.]

## 4.2 Testes funcionais

Isto vai ser uma tendência na nossa estratégia de testes, mas closed box foi a maneira preferida de testar visto que não queremos saber de como o teste foi implementado, mas sim se faz aquilo que esperamos.

## 4.3 Testes unitários

Mais uma vez o foco foi ter a certeza de que o código fazia tudo aquilo que queríamos, sem estarmos preocupados com a implementação, a implementação pode ser revista pelos membros do grupo.

## 4.4 Testes de sistemas e integração

Testes a API seguem o mesmo padrão onde o código interno dos serviços, as suas respostas e como elas são validadas não interessa, mas sim se o JSON e informação pretendida são enviados mediante o endpoint que foi acedido.