# Manual de Qualidade e Entrega Contínua

Projeto: **CloudIT**

Preparado por: André Brandão (84916), André Pedrosa (85098), Filipe Pires (85122), João Alegria (85048)

Data: 07/06/2019

**Conteúdo do relatório**

**Controlo de versões**

| Quando? | Responsável | Alterações significativas introduzidas |
|---|---|---|
| 16/04 | Filipe Pires | Contextualização e Gestão do Projeto (cap. 2) |
| 18/04 | Filipe Pires | Gestão do Projeto (cap. 2) e Testes (cap. 5) |
| 18/04 | André Pedrosa | Testes (cap. 5) |
| 18/04 | João Alegria | Gestão e Qualidade do Código (cap. 3) |
| 18/04 | André Brandão | CI & CD (cap.4) e Coding Style (cap. 3) |
| 22/04 | Filipe Pires | Acesso Rápido aos Recursos (cap. 1) |
| 25/04 | Filipe Pires | Atualizações pós-apresentação. |
| 07/06 | Filipe Pires | Atualizações entrega final. |

# 1- Accesso rápido aos recursos (*bookmarks* do projeto)

Sistematização dos links para os recursos desenvolvidos no projecto:

- Acesso ao(s) projecto(s) de código:
    a. Repositório GitHub
    b. Documentação da API de Integração (para programadores)
    c. Portainer
    d. Production Server
- Ambiente SQA :
    a. Análise Estática (SonarQube)
    b. Integração Contínua (Jenkins)
- Coordenação da equipa:
    a. User Stories (PivotalTracker)
    b. Documentação API (SwaggerHub)

# 2- Gestão do projeto

## Equipa e papéis

[Descrição dos papéis definidos na equipa e as responsabilidades de cada membro]
André Brandão – Team Manager
André Pedrosa – Product Manager
Filipe Pires – Documentation Manager
João Alegria – Quality Manager

## Gestão do *backlog* e atribuição de trabalho

**PivotalTracker Tqs19-104:**
https://www.pivotaltracker.com/n/projects/2324674
(atribuição de trabalho definida no PivotalTracker)

**Backlog:**
**1ª Iteração:**
"Introduction to the Platform", 4 user stories, 9 pontos
**2ª Iteração:**
"Job Posts", 3 user stories, 9 pontos
**3ª Iteração:**
"Navigation Inside the Platform", 3 user stories, 9 pontos
**4ª Iteração:**
"Communications Inside the Platform", 4 user stories, 10 pontos
**5ª Iteração:**
Resolução de bugs e adição de features opcionais
**Velocidade:** 9 pts / it

## *Issues tracking system*

Pretendemos seguir o do GitHub. Desta forma, sempre que um developer for de encontro a um issue, deverá identificá-lo no repositório de forma a que o progresso da solução seja facilmente acompanhado por todos.

# 3- Gestão e qualidade do código

## Guia para os colaboradores (*coding style)*

Documento dedicado ao estilo de código acessível através do seguinte link:
https://drive.google.com/open?id=1q9HuOPTXL6yYbKwjWPq7THRiKngCbkPx

## SCM *workflow*

O workflow que a nossa equipa decidiu adoptar é conhecido como Feature Branching. Achamos que este é a melhor estratégia, visto ser a mais dinâmica das existentes e a que melhor favorece trabalho autónomo e desacoplado. Desta forma cada um pode trabalhar independentemente numa nova feature sem se preocupar com o trabalho dos restantes, sendo apenas preciso criar um novo branch e fazer o desenvolvimento. Assim, cada user story terá um branch associado.
Com este workflow também temos ao nosso dispor várias vantagens como a de no ramo principal(master), se encontrar sempre uma solução funcional e possível de ser entregue a qualquer altura. Da mesma forma temos a possibilidade de reverter rápidamente qualquer erro que tenha passado por entre as várias camadas de garantia de qualidade, sendo apenas necessário reverter para a última versão funcional.

## Revisão de código

A revisão de código é um aspeto fundamental para o desenvolvimento do nosso produto, visto que tencionamos que cada adição ao nosso produto deve ser revista pelo menos por uma pessoa que nao tenha implementado a nova feature, sendo que deste modo há a garantia que de facto o novo código será correto e fará sentido ser adicionado para a melhoria da solução como um todo.
Para implementar isto tencionamos conjugar esta estratégia com o workflow adoptado, sendo que cada branch criado só será integrado com a solução principal através de um pull request, obrigando a que a nova feature seja revista ou por todos os elementos da equipa ou pelo menos um, sendo esses terceiros que validam e se necessário indicam o que está errado e o que deve ser melhorado.

## Análise estática

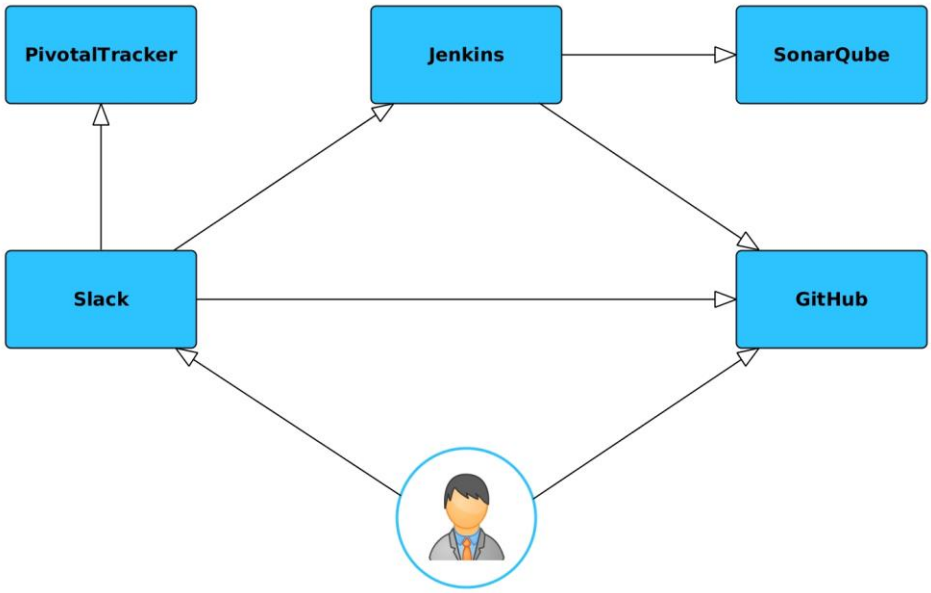
Para este ponto decidimos utilizar o SonaQube, ferramenta muito utilizada para este intuito, dando a possibilidade de fazer uma análise estática do código, verificando o estilo de escrita, erros sintáticos e até procurar padrões de programação, vendo se existem e estão bem implementados ou até dar sugestões de possivéis situações onde um padrão poderia melhorar o código feito.
Esta deteção de vulnerabilidades, bugs e más práticas de código será feita automáticamente atráves de mecanismos de integração e entrega contínua.

## Repositório de componentes [Opcional]

Apesar de optarmos por não usar, consideramos que utilizaríamos o Artifactory, partilhado por todos os grupos da turma.

# 4- Integração contínua & entrega contínua

Os developers irão desenvolver o código usando o Git para controlo de versões e gestão de código.
Estes irão discutir os vários problemas e soluções através do Slack, com o Slack também foi integrado plugins do Github, Jenkins e PivotalTracker para uma maior dinâmica e um conhecimento mais rápido sobre as alterações no projeto. Para automatizar a compilaçao e testes do sistema iremos usar o Jenkins integrado com o SonarQube para análise estática de código.
O nosso sistema será distribuído através de 3 containers docker:
- Um container com um servidor nginx para alojar o website.
- Um container para correr a REST API.
- Um container para alojar a base de dados.



# 5- Testes

## Testes funcionais/aceitação

Ferramenta: Selenium, Cucumber e Espresso
Heurísticas para o desenvolvimento de testes:

1. Desenvolver os testes segundo uma prespetiva de caixa fechada, em que a lógica interna da parte do sistema a ser testada não tem de ser conhecida pela pessoa que cria o teste.
2. Basear cada teste nos requisitos funcionais recolhidos, sendo escritos em formatos de stories e podendo cada uma ter diferentes cenários, e nas necessidades dos end-users da plataforma.
3. Retornar um resultado após a execução do teste que permita verificar se o sistema foi implementado da forma correta e de acordo com o que o cliente espera.

Boas práticas no uso de testes:

1. O conjunto dos testes deve garantir que os requisitos funcionais definidos são satisfeitos.
2. Devem ser executados durante os testes de sistema e de aceitação.

## Testes unitários

Ferramenta: JUnit e Mockito
Heurísticas para o desenvolvimento de testes:

1. Sempre que se cria um teste, providenciar: explicação curta do que se está a testar, a parte do software que se pretende testar (p.e. função ou classe), dados de teste para input ao teste, o resultado esperado do teste.
2. Não criar testes para tudo. Focar em testes que tenham impacto efetivo no comportamento do sistema.
3. Isolar o ambiente de desenvolvimento do de testes e usar dados de teste próximos dos de produção.
4. Isolar testes, garantindo a independência através do uso regular de objetos mock e serviços virtualizados.
5. Cobrir todos os possíveis caminhos de uma unidade. Tomar atenção às condições cíclicas.
6. Escrever teste unitário sempre que se encontra um defeito/erro e antes de o concertar.

Boas práticas no uso de testes:

1. Testes unitários devem ser corridos sempre que é feito um commit a alterações feitas ao repositório do source code, na máquina onde foi desenvolvido o código.
2. Antes de deixar que developers usem código de outros, garantir que os testes não falham.
3. Caso por alguma razão os problemas não possam ser concertados no momento, o responsável deverá alertar os restantes para o teste que falha e o motivo da falha se já se souber.

## Testes de sistema e de integração

Ferramentas: Spring Boot (embeded context tester)
Heurísticas para o desenvolvimento de testes:

1. Testar sempre partes individuais do sistema em grupo.
2. Procurar falhas que existam na integração entre unidades do sistema.
3. Desenvolver testes segundo uma perspectiva de caixa cinzenta, sendo que o developer tem de ter apenas algum conhecimento sobre como as várias unidades estão desenvolvidas internamente e conhecimento aprofundado sobre uma das unidades alvo do teste em questão.

Boas práticas no uso de testes:

3. Testes de integração devem ser executados assim que os testes unitários das unidades envolvidas passem.
4. A integração entre as várias unidades a testar deve estar bem definida antes de se realizar qualquer teste, de forma a que não tenham de ser alterados futuramente.

## Testes de desempenho

Ferramentas: Jmeter e Spring Boot.
Heurísticas para o desenvolvimento de testes:

1. Procurar pontos críticos no sistema que exijam um desempenho mínimo.
2. Procurar testar o sistema numa situação limite, mas que não exceda os limites do razoável.

Boas práticas no uso de testes:

1. Os testes de performance não devem ser executados constantemente.
2. Os testes não devem ser executados sobre o sistema em produção.
3. Tornar os testes e a publicação dos resultados o mais automatizados possíveis.