deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Relatório de Controlo de Qualidade

Francisco Fontinha [76490], João Vasconcelos [88808], Tiago Mendes [88886], Vasco Ramos [88931]

v2020-05-25

1. Bookmarks do Projeto

Sistematização dos links para os recursos desenvolvidos no projecto:

- Acesso ao(s) projecto(s) de código, bem como GitLab Agile e GitLab CI/CD:
    a. Repositório de Grupo (GitLab, mediante acesso)
- Ambiente de produção:
    a. Aplicação Web (Spring Boot)
    b. REST API (SpringBoot)
    c. Documentação da API (Swagger)
- Ambiente SQA :
    a. Análise estática (SonarQube)
    b. Monitorização (Nagios XI)
- Coordenação da equipa:
    a. Slack (aqui , mediante acesso)

2. Gestão do Projeto

2.1. Equipa e Papéis

Para facilitar a divisão de responsabilidades dentro da equipa, decidimos atribuir cargos específicos a cada elemento:

Cargo	**Descrição**	**Membro/s da equipa**
Product Owner	O product owner representa os interesses dos stakeholders. Tem um conhecimento detalhado sobre o produto e o domínio onde se insere e por isso os restantes elementos da equipa irão falar com ele para clarificar dúvidas sobre futuras funcionalidades do produto. Tem um papel ativo em aceitar os novos incrementos desenvolvidos para a solução final.	João Vasconcelos
Quality Engineer	Responsável, em articulação com outras funções, por promover as práticas de garantia de qualidade e colocar em prática instrumentos para medir a qualidade da implementação.	Francisco Fontinha
DevOps Master	O devops master é responsável pela infraestrutura de desenvolvimento e produção, aplicando as configurações necessárias à mesma. Gere a configuração e a preparação das máquinas de deployment, repositório git, infraestrutura da cloud,operações da base de dados, entre outras responsabilidades.	Tiago Mendes Vasco Ramos
Team Manager	O team manager assegura que existe uma divisão justa de tarefas e que o plano de desenvolvimento é seguido por	Vasco Ramos

	todos os elementos da equipa. Promove um bom ambiente dentro da equipa e assegura que os requisitos do projeto são entregues dentro dos prazos estabelecidos.	
Developer	O developer desenvolve a aplicação/projeto consoante o plano definido pelo team manager.	Francisco Fontinha João Vasconcelos Tiago Mendes Vasco Ramos

Tabela 1: Distribuição de Papéis de Trabalho dentro da Equipa

2.2. Gestão do Backlog e Atribuição de Trabalho

Tendo em conta que o desenvolvimento é orientado a user stories, existem diversas fases na criação e gestão do backlog:

1. **Adicionar as User Stories:** O Project Manager, tendo em conta as orientações do cliente, cria diversas user stories e adiciona-as ao backlog do projeto. Inicialmente, estas user stories não são atribuídas a nenhum elemento da equipa, nem têm uma timeline definida;
2. **Priorizar as User Stories:** O Project Manager estabelece a prioridade de cada story, através de um sistema de pontos;
3. **Estimar o tempo de desenvolvimento de cada User Story:** Numa reunião entre todos os elementos da equipa, estima-se o tempo que demorará a desenvolver cada story e complementa-se esta com informação extra (por exemplo: critérios de aceitação/qualidade);
4. **Desenvolver cada User Story:** O desenvolvimento das stories está pronto para ser iniciado. Assim, e tendo em conta os prazos apertados de desenvolvimento, cada story é atribuída a um developer. Após isto, os developers começam a desenvolver as user stories daquela interação, passando estas para o estado "*In Progress*";
5. **Entregar as User Stories:** Após o desenvolvimento de uma story, acompanhado dos respetivos testes, o developer entrega o seu trabalho. Isto vem acompanhado de um merge request, uma vez que cada feature será desenvolvida num branch diferente;
6. **Testar as User Stories:** Após o código entregue por um developer ser validado pelo ambiente de CI/CD, este é deployed num ambiente de teste. Após isto a story passará para o estado "*In Review*";
7. **Aceitar ou Recusar cada User Story:** O Project Manager, em conjunto com o cliente e com outros developers, testers e designers, verifica se os critérios de aceitação definidos inicialmente se verificam e, caso todas as entidades envolvidas concordem, a user story é aprovada, saindo do backlog para o painel "Closed". Caso contrário, esta story terá de ser refeita.

Neste projeto, a ferramenta de gestão de backlog que será utilizada será o GitLab que, levando a sua utilização ao máximo e tentando centralizar todo o desenvolvimento e processos numa só ferramenta, vai coexistir também como *Issue Tracking System*.

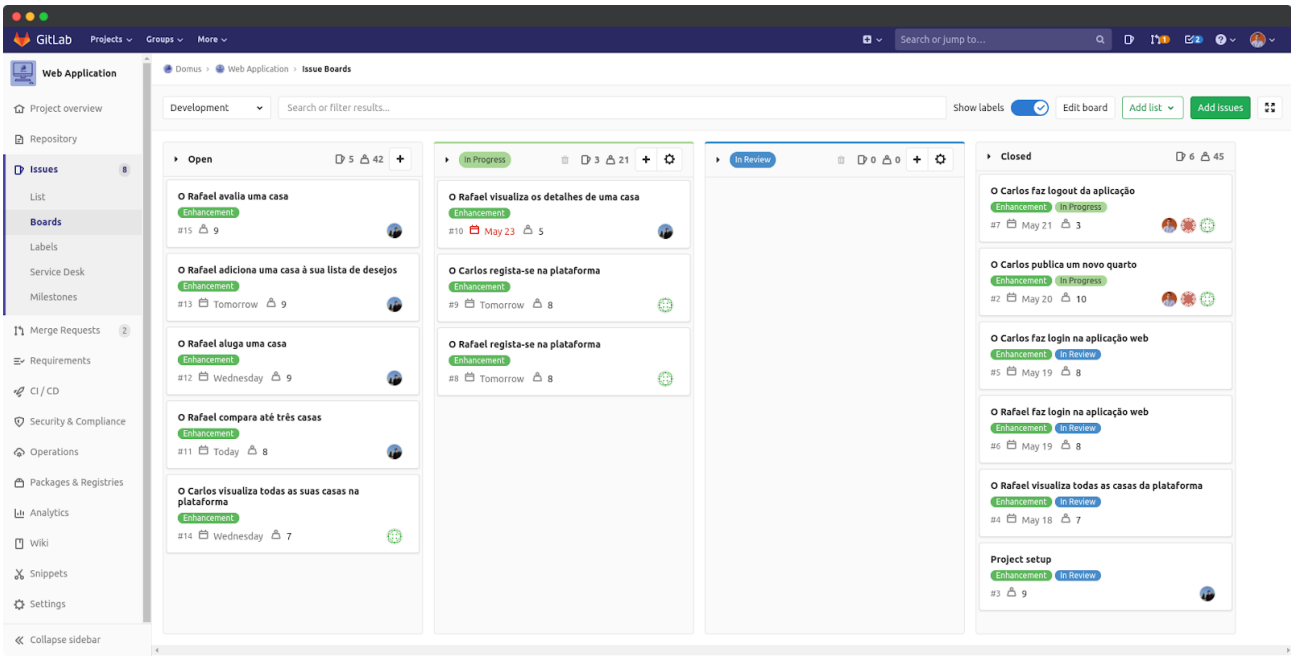

Figura 1: GitLab Agile - Gestão de Backlog + Issue Tracking System

3. Gestão e Qualidade do Código

3.1. Guia de Contribuição (Coding style)

Com o intuito de aplicar as melhores convenções possíveis referentes a escrita de código, iremos utilizar dois guias de estilo de código: um para a **linguagem Java** e outro para a **biblioteca ReactJS**. Em ambos os guias, estão definidas algumas regras e convenções da respetiva linguagem, bem como apresentados bons e maus exemplos práticos de utilização.

Relativamente à linguagem Java, iremos utilizar o seguinte guia de estilo de código, desenvolvido pela equipa do Android: https://source.android.com/setup/contribute/code-style.

Quanto à biblioteca ReactJS, o guia de estilo de código escolhido foi o seguinte: https://github.com/airbnb/javascript/tree/master/react.

3.2. Métricas de Qualidade de Código

A análise estática de código é um método para fazer o debugging do código, antes do programa ser corrido. Durante esta análise, compara-se o código escrito com regras gerais de escrita de código.

Para isto, recorremos ao SonarQube - uma plataforma *open source* para efetuar reviews automáticas e análise estática de código. Esta ferramenta, não só detecta código com bugs, como também bad smells e vulnerabilidades de segurança.

Para além disto, o SonarQube permite uma análise de código contínua, pelo que, para além de nos mostrar o estado da aplicação que estamos a desenvolver, também nos mostra quais os problemas introduzidos por cada incremento produzido.

Podemos, também, definir quality gates para o nosso projeto. Estas não são mais que meras métricas para a integração de novo código no projeto em desenvolvimento. Se uma nova submissão de código não passar numa quality gate, este código terá de ser refeito até que consiga atingir os padrões definidos. Ainda neste tópico, podemos definir quality gates relativos a:

- Duplicação de código
- Manutenção necessário para o código escrito
- Fiabilidade
- Segurança

A plataforma escolhida apresenta, também, um grande vantagem no que toca a integração contínua. Uma vez que o SonarQube é facilmente integrado com o GitLab CI/CD, é bastante simples automatizar esta análise estática de código, de forma a que a mesma seja executada de forma contínua. Por fim, tendo em conta que as quality gates default do SonarQube nos pareceram adequadas, decidimos manter estas ao invés de redefini-las para uma configuração própria.

(mostrar imagem do estado final do projecto bem como outra com os quality gates)

4. Pipeline de Entrega Continua (CI/CD)

4.1. Development workflow

4.1.1. SCM Workflow

Relativamente ao SCM Workflow, iremos usar **Git Feature Branch Workflow** do Bitbucket. Este subentende que:

- Sempre que um developer queira desenvolver uma nova feature/user story, este deverá fazer o seu desenvolvimento numa nova branch, especificamente criada para o efeito. Esta branch deve ser denominada de forma a que permita rapidamente identificar qual a feature/issue a ser tratada, utilizando o padrão **feature/<feature_name>**.
- Além disso, definimos que sempre que seja necessário corrigir algum bug devemos criar uma nova branch a partir do master, com o seguinte formato: **hotfix/<fix_name>**.
- Na nova branch criada, o developer edita e dá commits das suas implementações. Para além disto, este developer pode também dar push da sua branch para o repositório central, onde esta irá ser armazenado (backup).

- Quando um developer acabar de desenvolver as features associadas à branch que criou, este terá de criar um merge request para que a sua branch seja unida com a master branch. Desta forma, os outros membros da equipa irão receber uma notificação referente a esta situação.
- Os outros developers da equipa dão feedback sobre o código a ser inserido na master branch. Após isto, este código poderá ter de ser reformulado. Assim que o código for aprovado pelos reviewers, a branch onde está a nova feature será merged com a master branch.
- Por fim, define-se também (como acrescento ao flow em que nos baseámos) que todos os merge requests têm de ser aprovados por, pelo menos, um *reviewer* que não seja a próprio que submeteu o merge request.

Para mais detalhes, pode ser encontrada uma maior descrição do funcionamento deste workflow aqui.

4.1.2. Revisão de Código

De forma a melhorar a qualidade geral do código produzido, é necessário que este seja revisto por diversos developers, de forma a que se encontrarem erros e potenciais situações de riscos. Para que este processo decorra eficazmente, definimos um conjunto de princípios a seguir:

- Uma code review tem como objetivo uma análise minuciosa do código submetido. Tentar entender apenas algumas partes do código poderá, a longo prazo, ter consequências severas, pelo que é normal que uma code review demore uma elevada quantidade de tempo;
    - Caso alguma porção de código não seja percetível, o developer que o escreveu deverá reformular/explicar esta secção, sendo que os comentários não devem ser esquecidos;
    - Não se pode assumir que o código submetido funciona. É necessário fazer build do projeto e correr todos os testes associados ao mesmo. O reviewer poderá até criar novos testes;
    - Caso o código não tenha comentários explicativos, este deve ser documentado corretamente pelo developer que o escreveu;
    - É necessário rever, também, o código “temporário”, uma vez que este se poderá tornar em código para produção;
    - Deve ser realizada uma review, quer aos testes, quer aos *build files* associados a código que está a ser revisto;
    - As reviews de código devem ter em atenção se o code style está de acordo com o definido no início do projeto;
    - A arquitetura de uma solução poderá, também esta, ser revista;
    - Os comentários de uma code review devem ser críticas construtivas;
    - Ao fazer uma code review, as sugestões devem ser feitas de acordo com a seguinte prioridade:

- Melhorias a nível funcional;
- Alterações para manter o código clean e fácil de manter;
- Por fim, sugestões para otimizar o código.

- Acompanhar o estado de uma code review é tão importante como fazer a code review, pelo que cada developer deverá fazer o follow up das reviews que fez.

Relativamente ao processo de code review implementado neste projeto, este tem como suporte as ferramentas disponibilizadas pelo GitLab. Sempre que é feito um merge request, inicia-se um processo de code review do código submetido.

Por fim, para esclarecimento futuro, uma user story é considerada como terminada após completar todo este processo de: Merge Request -> Build e Execução de Testes Automáticos -> Code Review e Validação de Código -> Deployment. Ou seja, uma user story é concluída quando esta é deployed com sucesso, respeitando o flow previamente especificado.

4.2. Ferramentas e Pipelines de CI/CD

[Description of the practices defined in the project for the continuous integration of increments and associated resources. Provide details on the tools setup and config.]

- Incluir pipelines
- Integração com o slack
- Integração própria com o nosso serviço

[Description of practices for continuous delivery, likely to be based on *containers*]

4.3. Repositório de Artifacts e Containers

Para facilitar e agilizar a gestão de *artifacts*, bem como a execução das pipelines de integração e *deployment*, decidimos usar o GitLab Registry (para Maven *artifacts* e Docker *containers*). Estes permitem-nos uma rapidez maior na execução das pipelines de integração através dos mecanismos de cache do GitLab, bem como rapidamente executar o deployment, tirando partido dos docker containers respetivos aos vários serviço que são criados durante as pipelines de execução. O repositório-mãe encontra-se nos seguinte endereços:

- Package Registry: https://gitlab.com/groups/tqs-domus/-/packages
- Container Registry: https://gitlab.com/groups/tqs-domus/-/container_registries.

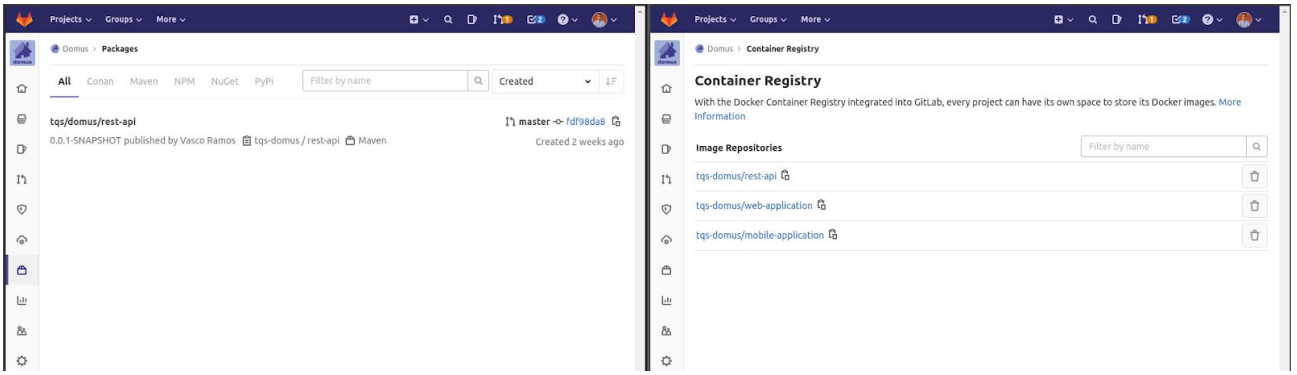

Figura X: GitLab Registry - Maven Artifacts + Docker Containers

5. Testes de Software

5.1. Estratégia de Teste

Esta secção é destinada a documentar a estratégia de testes usada no projeto. Desde o início do desenvolvimento, que o objetivo foi sempre aplicar TDD (*Test-Driven Development*), de forma a que se crie apenas o essencial para implementar os testes e, a partir daí, desenvolver as funcionalidades.

Tecnologicamente, usámos JUnit 5 e Mockito (quando necessário) para os testes da REST API, Jest e Puppeteer para os testes funcionais na Web App e Jest para testar a Mobile App.

5.2. Testes Funcionais/Aceitação

Este tipo de testes são, geralmente, realizados em cooperação com o cliente (interface), uma vez que são os testes que irão validar se a aplicação está de acordo com o que o cliente pretende, que caso esteja, significa que a aplicação está pronta para lançamento. Para este tipo de teste, usamos casos de teste que cobrem os cenários típicos sob os quais esperamos que o software seja usado. Estes testes devem ser conduzidos num ambiente de "produção", e num hardware que seja igual ou próximo do que cliente usará.

Este tipo de testes são, também conhecidos como *black-box testing*, pois não existe qualquer noção do sistema interno da aplicação, sendo apenas enviado os valores de entrada e analisado os valores de saída correspondentes.

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

5.3. Testes Unitários

De forma a garantir que todas as funcionalidades, por mais minimalista que sejam, não perturbem o bom funcionamento da aplicação e funcionam como pretendido, são expostas a testes unitários. Cada teste unitário executa o código fonte de uma e apenas uma funcionalidade, daí a ser unitário. Tem como metodologia *white-box testing*, ou seja, tem como objetivo testar a estrutura interna de uma aplicação.

```java
@Test
void testCreateHouseReview_missingParameters() {
  UserDTO userDTO = new UserDTO("v1@ua.pt", "Vasco", "Ramos", "pwd", "123", "M", null);
  User user = new ModelMapper().map(userDTO, User.class);
  Locatario locatario = new Locatario();
  locatario.setUser(user);

  when(locatarioRepository.findById(anyLong())).thenReturn(Optional.of(locatario));

  HouseDTO houseDTO = new HouseDTO("Av. da Misericórdia", "São João da Madeira", "3700-191",2, 2, 2, 300,
                                   true, 230, "Casa T2","Casa muito bonita", "WI-FI;Máquina de lavar",
                                   null, null);
  House house = new ModelMapper().map(houseDTO, House.class);

  when(repository.findById(anyLong())).thenReturn(Optional.of(house));

  HouseReviewDTO reviewDTO = new HouseReviewDTO(0L, 0L, "string", 0D);

  when(reviewRepository.save(any(HouseReview.class))).thenThrow(ConstraintViolationException.class);

  assertThrows(ErrorDetails.class, () -> {
    service.registerReview(reviewDTO);
  });
}
```

Figura X: HouseService Unit Testing - Create House Review operation

5.4. Testes de Sistema e Integração

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

API testing

5.5. Testes de Desempenho [Opcional]

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

6. Monitorização do Ambiente de Produção

Tendo em mente a necessidade de monitorizar e consultar o estado das máquinas usadas no ambiente de produção, utilizamos a ferramenta Nagios XI, uma extensão à interface já existente do Nagios Core, com a função de monitorizar sistemas, redes e infraestruturas.

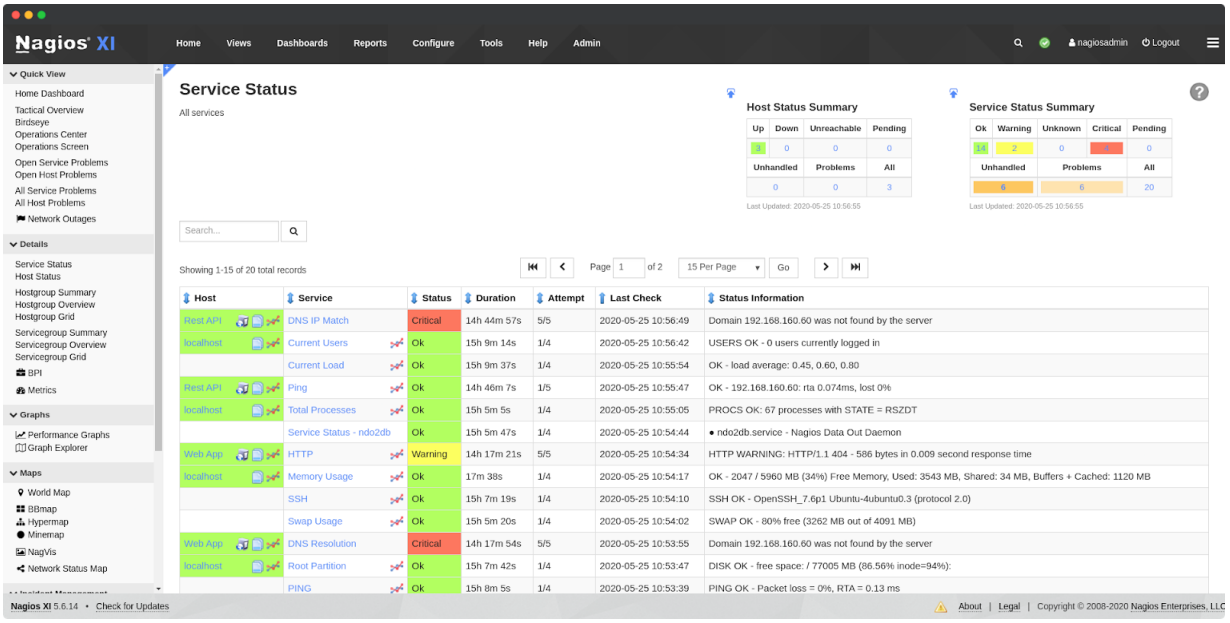

Figura X: Dashboard do Nagios XI

Posto isto, na nossa dashboard do Nagios XI, podemos monitorizar os hosts configurados (localhost e deployment - REST API + Web App), e os serviços a eles associados como, por exemplo, o uso de memória, a carga atual e http, entre outros. Para além disto, é, também, possível visualizar os dados processados sobre o desempenho através de gráficos, como podemos observar na imagem abaixo.

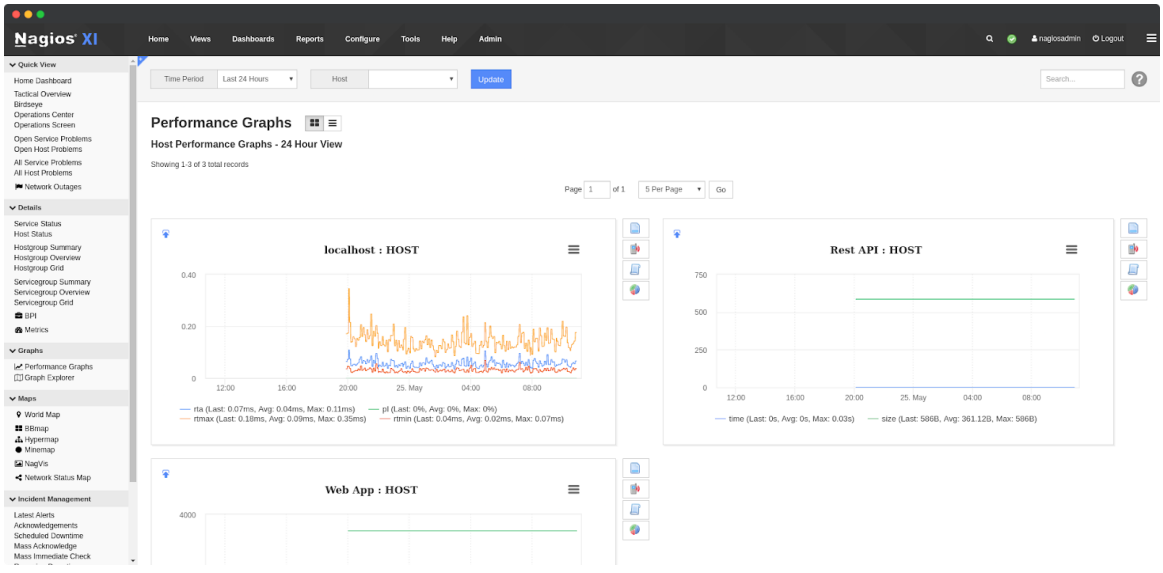

Figura X: Gráficos de desempenho do Nagios XI