TQS: Quality Assurance manual

1 Project management

1.1 Team and roles

Role	Name	Email	Nmec
Team Manager	Airton Moreira	Agm@ua.pt	100480
Product Owner	João Diogo Craveiro Ferreira	jd.ferreira@ua.pt	95316
QA Engineer	João Diogo Craveiro Ferreira	jd.ferreira@ua.pt	95316
DevOps Master	Airton Moreira	Agm@ua.pt	100480

1.2 Agile backlog management and work assignment

No âmbito do projeto *ElectraNet*, adotamos uma metodologia ágil baseada em sprints semanais, com a gestão do backlog centralizada no Jira. No início de cada sprint, a equipa define colaborativamente as *user stories* a desenvolver, sendo posteriormente distribuídas pelo Gestor de Equipa de forma equilibrada.

Cada *user story* é desdobrada em subtarefas específicas de desenvolvimento e testes, assegurando uma implementação estruturada e orientada para a qualidade. A integração com Xray permite-nos monitorizar automaticamente os resultados dos testes, mantendo a rastreabilidade e visibilidade no Jira.

// TO DO --- Put jira user stories coverage here (pics)

//

2. Code quality management

2.1 Guidelines for contributors (coding style)

Todas as contribuições para o repositório de código do projeto *ElectraNet* devem seguir as seguintes diretrizes:

- Os nomes das classes devem utilizar PascalCase (por exemplo: HelloWorld.java);
- Os nomes dos *packages* devem ser declarados no início do ficheiro, em minúsculas e sem números;
- Devem ser utilizados *imports* específicos, evitando o uso de *;
- A declaração de variáveis deve seguir a convenção camelCase (por exemplo: someVariable);
- O código deve seguir uma formatação consistente, recorrendo à ferramenta Prettier;
- O âmbito das variáveis deve ser o mais restrito possível;
- Utilização do Lombok para geração automática de construtores, *getters*, *setters* e métodos toString;
- O código deve ser devidamente comentado, de forma a garantir clareza e facilidade de manutenção;
- Todas as contribuições devem incluir testes automatizados utilizando o framework JUnit;
- A análise contínua da qualidade do código é realizada com recurso ao SonarQube integrado no IntelliJ IDEA, promovendo a deteção precoce de *code smells* e potenciais vulnerabilidades.

2.2 Code quality metrics and dashboards

A análise estática do código no projeto *ElectraNet* é realizada através do SonarQube, utilizando a *Quality Gate* padrão "Sonar Way", a qual se revelou adequada para os nossos objetivos de qualidade sem necessidade de personalização adicional.

Esta *Quality Gate* impõe os seguintes critérios, que devem ser cumpridos para validação do código:

- Ausência de novos *bugs*;
- Ausência de novas vulnerabilidades;
- Dívida técnica limitada no novo código;
- Todos os *security hotspots* devem ser revistos;
- Cobertura de testes igual ou superior a 80%;
- Percentagem de linhas duplicadas inferior ou igual a 3%.

A análise de qualidade é automaticamente desencadeada sempre que é efetuado um *pull request* ou um *push* para a *main branch*, sendo os resultados publicados no painel do SonarQube integrado no IntelliJ IDEA. Este processo garante visibilidade contínua sobre a saúde do código e facilita a deteção precoce de problemas.

3. Continuous delivery pipeline (CI/CD)

3.1 Development workflow

No projeto *ElectraNet*, adotamos o fluxo de trabalho **GitHub Flow** para o desenvolvimento. Para cada tarefa registada na plataforma Jira, é criada uma *branch* dedicada no repositório Git. Após a conclusão da implementação, é submetido um *Pull Request* para a *main branch* do respetivo submódulo.

Este *Pull Request* deve ser revisto por pelo menos um outro membro da equipa, garantindo a conformidade com os padrões de qualidade definidos e a aprovação na análise automática de código efetuada pelo SonarQube.

Atualmente, encontra-se implementada a fase de **Integração Contínua (CI)**, a qual executa automaticamente testes e validações após cada *push* ou *pull request*, assegurando a estabilidade e qualidade do código integrado.

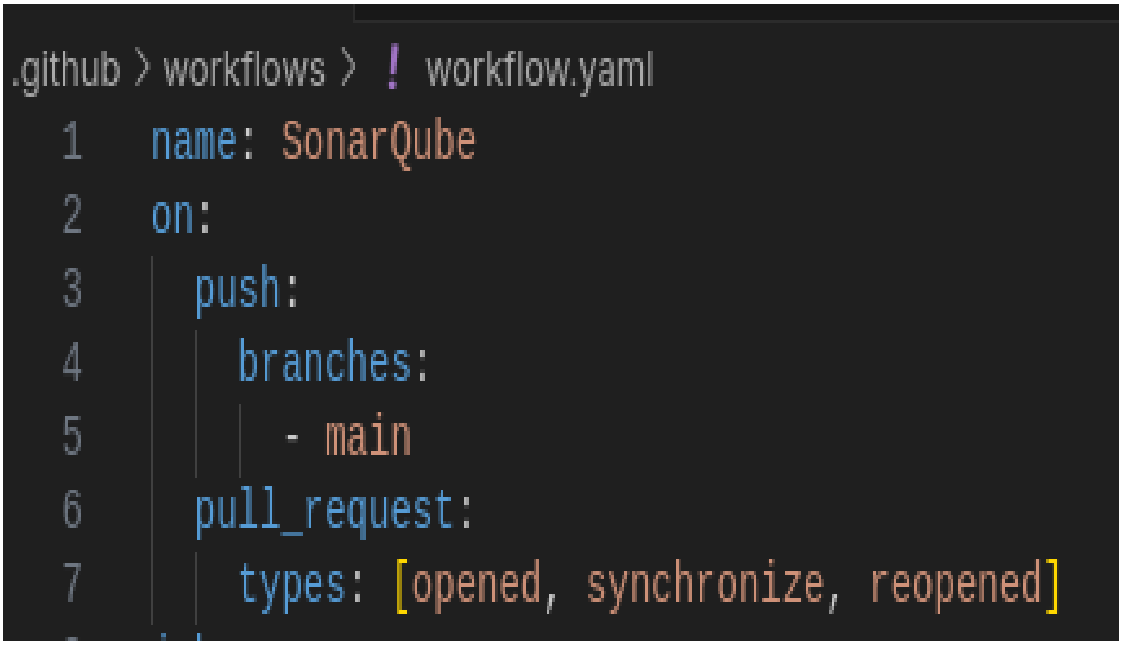
A fase de **Entrega Contínua (CD)** será integrada futuramente, com o objetivo de permitir a disponibilização automática das versões validadas em ambiente de staging ou produção.

3.2 CI/CD pipeline and tools

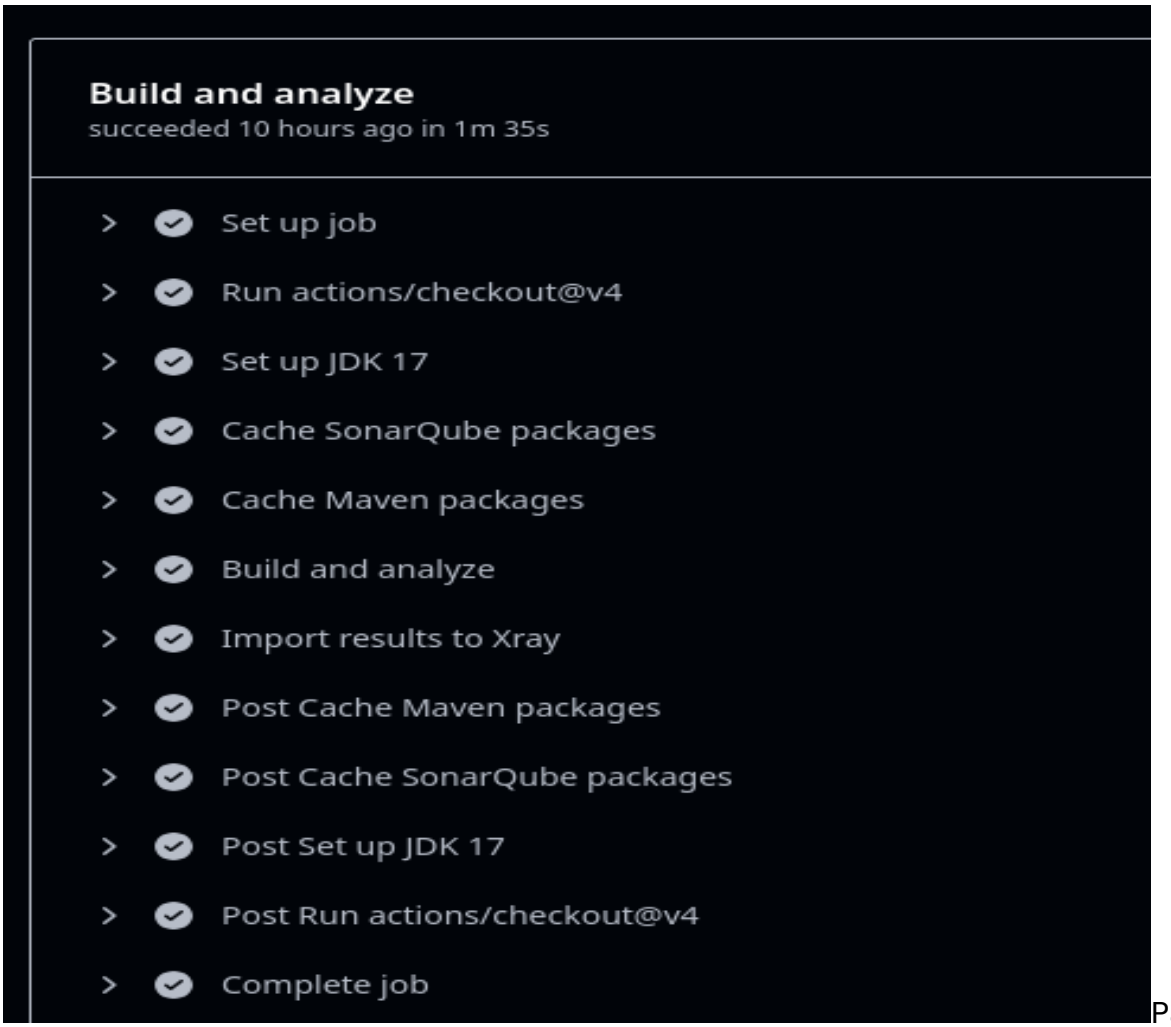
A pipeline de Integração Contínua (CI) implementada no projeto *ElectraNet* visa detetar precocemente erros e garantir a estabilidade contínua da aplicação ao longo do seu desenvolvimento.

Recorremos ao **GitHub Actions** para automatizar este processo. A cada *commit* ou *pull request* realizado no repositório principal, o código é automaticamente testado e validado, assegurando que novas alterações não comprometem a integridade do sistema.

A arquitetura do projeto assenta na **containerização** dos seus módulos, o que permite o desenvolvimento e validação de componentes de forma isolada. Esta abordagem favorece a escalabilidade e prepara o projeto para uma futura implementação de uma pipeline de Entrega Contínua (CD), permitindo testar e atualizar serviços específicos sem afetar o sistema global.




```
  SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=APG2000_TqsTeam2025
```



Para além da execução automática de testes em cada *pull request* ou *push*, o projeto *ElectraNet* integra uma **GitHub Action** que permite o envio dos resultados dos testes automatizados para o **Jira**, a nossa plataforma de gestão de tarefas. Esta integração reforça a rastreabilidade entre o código desenvolvido e os requisitos definidos, facilitando o acompanhamento da qualidade e do progresso das *user stories*.

```yaml
- name: Import results to Xray
  uses: mikepenz/xray-action@v3
  with:
    username: ${{ secrets.XRAY_CLIENT_ID }}
    password: ${{ secrets.XRAY_CLIENT_SECRET }}
    testFormat: "junit"
    testPaths: "server/eletranet/backend/target/surefire-reports/TEST-*.xml"
    testExecKey: "SCRUM-48"
    projectKey: "SCRUM"
```

3.3 System observability

3.4 Artifacts repository

4 Software testing

4.1 Overall strategy for testing

No projeto ElectraNet, a estratégia de desenvolvimento variou conforme o serviço em causa. No backend, os testes unitários vão ser desenvolvidos em paralelo com as funcionalidades, utilizando ferramentas específicas para testar controladores, serviços e repositórios de forma eficiente.

No frontend, os testes vão ser criados após a implementação das funcionalidades principais, permitindo uma rápida identificação e correção de erros nos fluxos de utilizador sem necessidade de reescrita dos testes. Ferramentas como Cucumber BDD e REST-Assured vão ser usadas para simular estes fluxos de forma rigorosa.

O pipeline de Integração Contínua incorpora métricas de qualidade de código que bloqueiam merges na branch principal caso os critérios do Quality Gate não sejam cumpridos, garantindo a qualidade e estabilidade das alterações antes da produção.

Esta abordagem assegura que o ElectraNet mantém elevados padrões de qualidade, minimizando riscos e facilitando a manutenção e evolução da plataforma.

4.2 Functional testing/acceptance

4.3 Unit tests

Paralelamente ao desenvolvimento das funcionalidades, são também desenvolvidos testes unitários para garantir que as funcionalidades específicas adicionadas por estes componentes funcionam corretamente.

Estes testes são desenvolvidos utilizando o framework JUnit 5, com o apoio do framework Mockito para criação de mocks.

vao ser desenvolvidos testes unitários para as várias camadas da aplicação Spring Boot (controladores, serviços e repositórios),

4.4 System and integration testing

4.5 Performance testing