DeCA|ua | departamento de comunicação e arte universidade de aveiro

universidade de aveiro

Relatório Mini Projeto M1A

Tecnologias e Desenvolvimento Web

Vânia Morais
102383
Mestrado em Comunicação e Tecnologias Web

Índice

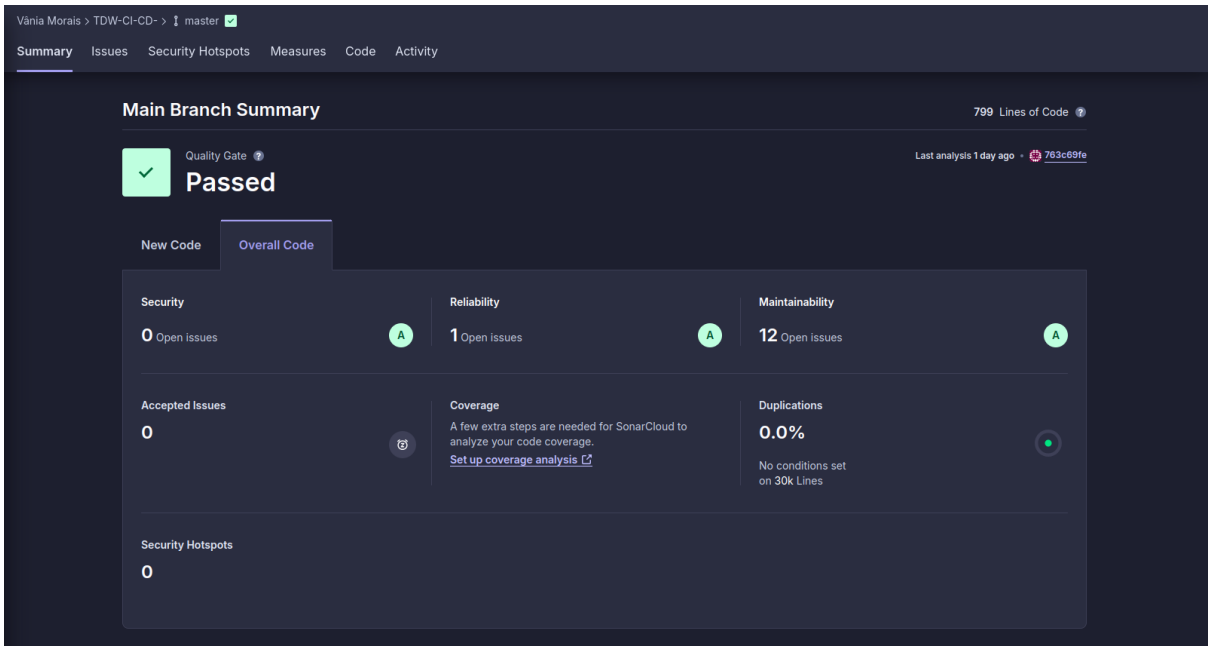
Introdução

O objetivo deste projeto foi implementar uma pipeline CI/CD para automatizar testes, integração e validação de qualidade num projeto simples. Usando um template fornecido, foi integrado o Contentful como CMS (Content Management System) e foram realizados testes com Jest e SonarCloud. A pipeline foi desenvolvida em paralelo no GitHub e no GitLab, garantindo consistência e automatização na gestão do projeto.

Integração de Contentful e Testes

Neste projeto, o Contentful foi integrado como sistema de gestão de conteúdo (CMS) para armazenar e organizar dados dinâmicos, como posts e autores, permitindo atualizações de conteúdo sem a necessidade de novas implementações de código. A API do Contentful é usada para obter os dados diretamente na aplicação Next.js, sincronizando o conteúdo com a pipeline sempre que o CMS é atualizado.

Os testes foram configurados com Jest. Existem 6 ficheiros, nos quais é testado a renderização correta de imagens, a formatação das datas, exibição correta dos posts e se as funções que obtêm os dados funcionam corretamente. Por vezes é usado o *mock* para simular algum dado necessário ao teste.



Adicionalmente, o SonarCloud foi integrado para análise de qualidade e segurança do código, identificando potenciais bugs e garantindo a conformidade com boas práticas.

Inicialmente foi também pensado usar o Selenium para testar o frontend, mas após alguma pesquisa, foi percebido que seria complicado aplicar este tipo de teste na pipeline.

Desenvolvimento da Pipeline CI/CD no GitHub

De modo a garantir um fluxo de integração contínua e entrega contínua (CI/CD), foi implementada uma pipeline no GitHub Actions, estruturada para otimizar a automação e qualidade do projeto.

```yaml
name: Node.js CI

on:
  push:
    branches:
      - '**' # Executa a pipeline em todos os commits em todas as branches
  schedule:
    - cron: "0 0 * * 1-5" # Executa às 00:00 de todos os dias de segunda a sexta
  workflow_dispatch: # Permite que a pipeline seja acionada manualmente
  repository_dispatch: # Webhook para quando o CMS for atualizado
    types:
      - cms-updated
```

Neste excerto, o código faz com que a pipeline seja acionada manualmente pela interface do GitHub, sempre que um commit é feito, de segunda à sexta às 00h00 (foram escolhidos estes dias da semana para não exceder os limites do github) e cada vez que o CMS é atualizado.

```yaml
jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      # Cache Node.js modules
      - name: Cache Node.js modules
        uses: actions/cache@v2
        with:
          path: ~/.npm
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-node-

      # Checkout do repositório
      - name: Checkout repository
        uses: actions/checkout@v2

      # Configuração do Node.js
      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '20'
```

Aqui, é definido o ambiente onde a pipeline vai ser executada (ubuntu). É também armazenado em cache os pacotes do Node.js, de modo a ser mais rápido correr a

pipeline futuramente. A cache é atualizada sempre que forem detetadas novas dependências. O repositório é clonado através do checkout, de modo a ter o código disponível para os próximos passos. Por fim, é configurado o ambiente Node.js.

```yaml
# Instalar dependências
- name: Install dependencies
  run: |
    cd mp1-contentfull-tdw
    npm install

# Usar o lint
- name: Run lint
  run: |
        cd mp1-contentfull-tdw
        npm run lint

# Usar o prettier
- name: Run prettier
  run: |
        cd mp1-contentfull-tdw
        npm run prettier
```

Neste trecho, são instaladas as dependências do projeto e é aplicado o *Lint* e o *Prettier* ao código. O *Lint* analisa o código e identifica erros de sintaxe e outras inconsistências. Como grande parte do código já veio feito do template, foi necessário ignorar algumas regras no ficheiro *eslint.config.mjs*. Já o *Prettier* formata o código de modo a ser visualmente agradável, de fácil leitura e manutenção.

```yaml
# Executar os testes
- name: Run tests
  run: |
    cd mp1-contentfull-tdw
    npm test

- name: Build the project
  run: |
    cd mp1-contentfull-tdw
    npm run build
  env:
    CONTENTFUL_SPACE_ID: ${{ secrets.CONTENTFUL_SPACE_ID }}
    CONTENTFUL_ACCESS_TOKEN: ${{ secrets.CONTENTFUL_ACCESS_TOKEN }}
    CONTENTFUL_PREVIEW_ACCESS_TOKEN: ${{ secrets.CONTENTFUL_PREVIEW_ACCESS_TOKEN }}
    CONTENTFUL_PREVIEW_SECRET: ${{ secrets.CONTENTFUL_PREVIEW_SECRET }}
    CONTENTFUL_REVALIDATE_SECRET: ${{ secrets.CONTENTFUL_REVALIDATE_SECRET }}
  if: success()

- name: Deploy to Netlify
  run: |
    cd mp1-contentfull-tdw
    npx netlify-cli deploy --site ${{ secrets.NETLIFY_SITE_ID }} --auth ${{ secrets.NETLIFY_AUTH_TOKEN }} --prod
  if: success()
```

Por fim, são corridos os testes falados anteriormente e é feito o build e o deploy do projeto.
Para que o build funcionasse, foi necessário fornecer todas as variáveis ambiente que tinha localmente no ficheiro *.env.local*, de modo a ser possível aceder aos dados do CMS. No final procedeu-se ao deploy do projeto no Netlify. É de notar que caso o build não funcione, não se avança para o deploy.

Desenvolvimento da Pipeline CI/CD no GitLab

A segunda plataforma escolhida para replicar a pipeline foi o GitLab, que oferece uma interface que permite configurar todos os aspetos do CI/CD diretamente na sua própria plataforma. Embora essa integração seja vantajosa, a experiência revelou algumas dificuldades, como a visualização em tempo real do progresso da pipeline e a identificação dos erros que impediam a execução. A pipeline no GitLab foi dividida em 'stages' (etapas) e as credenciais sensíveis foram geridas através de variáveis de ambiente, chamadas de 'variáveis' no GitLab.

```yaml
stages:
  - test
  - build

image: node:20

variables:
  CONTENTFUL_SPACE_ID: $CONTENTFUL_SPACE_ID
  CONTENTFUL_ACCESS_TOKEN: $CONTENTFUL_ACCESS_TOKEN
  CONTENTFUL_PREVIEW_ACCESS_TOKEN: $CONTENTFUL_PREVIEW_ACCESS_TOKEN
  CONTENTFUL_PREVIEW_SECRET: $CONTENTFUL_PREVIEW_SECRET
  CONTENTFUL_REVALIDATE_SECRET: $CONTENTFUL_REVALIDATE_SECRET

cache:
  key: ${CI_COMMIT_REF_SLUG}
  paths:
    - node_modules/

before_script:
  - cd mp1-contentfull-tdw
  - npm install

run_tests:
  stage: test
  script:
    - npm run lint
    - npm run prettier
    - npm test
  rules:
    - if: '$CI_COMMIT_BRANCH == "master"'

build:
  stage: build
  script:
    - npm run build
  rules:
    - if: '$CI_COMMIT_BRANCH == "master"'
```

Nesta pipeline são definidos os stages: test e build. Não é feito o deploy, uma vez que o *auth_token* do Netlify já não estava guardado aquando a criação desta pipeline. São definidas as variáveis ambiente e é configurada a cache para acelerar as instalações das dependências nas execuções futuras. Na parte *before_script* são definidos os comandos a serem executados antes da cada etapa, sendo eles mudar para o diretório onde está o projeto e instalar as dependências do mesmo. Depois, na etapa *test* são executados o *Lint*, o *Prettier* e os testes definidos no projeto. Por fim, o projeto é compilado. É de notar que as etapas descritas anteriormente apenas correm na branch master.

Desafios e Soluções

Durante este projeto houve alguns desafios. O primeiro foi fazer com que o código fosse corretamente traduzido para JavaScript para ser possível fazer testes em *Jest*. Mesmo usando *Babel* ainda foram encontradas algumas dificuldades, que após muita pesquisa, foram resolvidas.
Mais tarde, houve dificuldades a dar build ao projeto, supostamente devido ao *path* de duas pastas. Depois de alguma pesquisa, foi testado se o projeto funcionaria sem estas pastas, o que resultou e o build passou a decorrer sem qualquer problema.
Já na parte das pipelines, houve dificuldade a dar build no GitHub, uma vez que não me tinha apercebido que era necessário escrever as variáveis do Contentful.
Por fim, houve dificuldade no GitLab, principalmente em perceber como a interface funcionava e as diferenças na pipeline do GitHub para o GitLab.

Conclusão

Este projeto permitiu implementar uma pipeline CI/CD funcional tanto no GitHub como no GitLab, garantindo automação e qualidade contínua num projeto simples. Os desafios enfrentados, como a adaptação do código para Jest e o ajuste de variáveis do Contentful, fortaleceram a compreensão dos processos de integração e entrega contínua, além de melhorarem as competências técnicas. Esta experiência proporcionou uma visão prática das diferenças entre plataformas e consolidou uma base sólida para projetos futuros com pipelines mais complexas.