|  |
| --- |
| **GTI**  **SPRINT 2 – MISSÃO 7**  **PROJETO: “DEPLOYMENT QUALITY ASSURANCE”** |
| **ESTUDO DE CASO**  Uma certa empresa decidiu estabelecer uma cultura QUALITY ASSURANCE em seu modelo de negócio, visando impactar positivamente processos de qualidade em suas as suas áreas de operação e tecnologia.  **ESCOPO DO PROJETO**  O projeto será composto por 3 Sprints que se complementam, onde os alunos deverão construir ações que validem a empresa a possuir uma cultura orientada a Q.A.  Em **duplas** os alunos desenvolverão projeto 3 em Sprints:   * SPRINT 1: Vale 0,5 ponto na AC-1 e presenças nas aulas * **SPRINT 2: Vale 1 ponto na AC-2 e presenças nas aulas** * SPRINT 3: Vale 1 ponto na AC-3 e presenças nas aulas   **OBJETIVO**  Aprender as nuances e aplicabilidade do *QUALITY ASSURANCE* em uma organização. Construir um projeto de implementação de Gerenciamento de Qualidade Total e realizar atividades que valem nota.  **SPRINT 2 (1 ponto)**  Início: **18/09** – Término: **09/10**. Vale 1,0 ponto na AC-2 e presenças nas aulas.  Composto por 4 missões que se complementam para a entrega total do projeto:   * Missão 5: Automação de Testes I– Vale 25% da AC-2 * **Missão 6: Automação de Testes II – Vale 25% da AC-2** * **Missão 7: Automação de Testes II – Vale 25% da AC-2** * Missão 8: Testes de API /QA em Mobile e entrega final – Vale 25% da AC-2   **MISSÃO 7**  **VALE 25% DA NOTA AC-2**  **Objetivo:** Construir dois cenários de integração CI/CD em código Python com o GitHub Actions.  **Resultado Esperado:**   * 1: Após o push, a pipeline roda os testes unitários e valida o código. Alterações no código acionam o CI/CD. * 2: Mudanças que quebram o código são identificadas automaticamente, e os alunos podem visualizar como o CI/CD reage a erros. * 3: Automatização completa com versionamento automático.   **TAREFA 1 – PREPARAÇÃO:**   1. Baixe o arquivo esse “**Missão7-Projeto QA - ADS-5.pdf**” disponível no AVA; 2. Abra o GitHub oficial da dupla/trio e em seguida abra o repositório que estão usando para o projeto; 3. Suba no seu repositório o arquivo “**Missão7-Projeto QA - ADS-5.pdf**”; 4. Agora abra o projeto deste repositório e visualize o quadro Kanban que está gerenciando o projeto; 5. Criar e colocar o cartão MISSÃO 7 para a lista EM ANDAMENTO;   **TAREFA 2 – EXECUÇÃO DOS CENÁRIOS**  **Cenário 1: CI/CD para um Projeto Python com Testes Unitários**  Passo 1: Abra seu repositório no GitHub  Abra o VS Code, abra um novo terminal e clone o repositório:  **git clone https://github.com/<seu-usuario>/<seu-repositorio>/**  **cd <seu-repositorio>/**  **Passo 2: Criar o Projeto Python**  Dentro do repositório clonado, crie uma pasta chamada **src** para armazenar seu código.  **mkdir src**  Crie um arquivo **main.py** no diretório src com o seguinte conteúdo:  ***def soma(a, b):***  ***return a + b***  ***if \_\_name\_\_ == "\_\_main\_\_":***  ***resultado = soma(3, 5)***  ***print(f"O resultado é {resultado}")***  **Passo 3: Criar Testes Unitários**  Dentro do repositório clonado, crie uma pasta chamada **tests** dentro do repositório.  **mkdir tests**  Dentro de tests, crie um arquivo chamado test\_main.py com o conteúdo:  ***import unittest***  ***from src.main import soma***  ***class TestMain(unittest.TestCase):***  ***def test\_soma(self):***  ***self.assertEqual(soma(3, 5), 8)***  ***if \_\_name\_\_ == '\_\_main\_\_':***  ***unittest.main()***  Execute o teste localmente para garantir que tudo está funcionando:  **python -m unittest tests/test\_main.py**  **Passo 4: Configurar GitHub Actions**  No repositório, ainda do VSCODE, crie a pasta .github/workflows dentro do repositório.  Entre no diretório e crie um arquivo ci.yml dentro dessa pasta com o seguinte conteúdo:  O arquivo deve se chamar ci.yml  Coloque o código abaixo:  ***name: Python CI***  ***on:***  ***push:***  ***branches: [ main ]***  ***pull\_request:***  ***branches: [ main ]***  ***jobs:***  ***build:***  ***runs-on: ubuntu-latest***  ***steps:***  ***- uses: actions/checkout@v2***  ***- name: Set up Python***  ***uses: actions/setup-python@v2***  ***with:***  ***python-version: '3.x'***  ***- name: Install dependencies***  ***run: |***  ***python -m pip install --upgrade pip***  ***pip install unittest***  ***- name: Run tests***  ***run: |***  ***python -m unittest discover tests***  **Passo 5: Commit e Push no GitHub**  Volte ao VSCODE e adicione as alterações ao Git e faça commit:  ***git add .***  ***git commit -m "Adiciona teste unitário e configuração do GitHub Actions"***  ***git push origin main***  Após o push, o GitHub Actions automaticamente iniciará a pipeline, rodando os testes no ambiente configurado.  **Passo 6: Verificar o Status no GitHub**  Acesse o repositório no GitHub, vá até "Actions" e veja a execução da pipeline. Se tudo estiver correto, verá um "tick verde" indicando que os testes passaram.  **Cenário 2: Alteração no Código e CI/CD Automatizado**  **Passo 7: Modificar o Código e Verificar o CI/CD**  Modifique o arquivo src/main.py, alterando a função soma para:  ***def soma(a, b):***  ***return a + b + 1 # Introduzindo um erro proposital***  Commit e Push das alterações:  ***git add .***  ***git commit -m "Modifica a função soma"***  ***git push origin main***  O GitHub Actions será acionado novamente e, desta vez, o teste deve falhar, já que o valor esperado mudou.  **Passo 8: Corrigir o Erro e Revalidar**  Corrija o erro no código:  ***def soma(a, b):***  ***return a + b # Corrigindo o erro***  Faça o commit da correção:  **git add .**  **git commit -m "Corrige a função soma"**  **git push origin main**  A pipeline será executada novamente, agora com os testes passando.  **Cenário 3: CI/CD com Versionamento Automático**  **Passo 9: Adicionar Versionamento ao Projeto**  Crie um arquivo version.txt no repositório com a versão 0.1.0.  Modifique o ci.yml para fazer o incremento da versão automaticamente em cada push:  ***name: Python CI***  ***on:***  ***push:***  ***branches: [ main ]***  ***pull\_request:***  ***branches: [ main ]***  ***jobs:***  ***build:***  ***runs-on: ubuntu-latest***  ***steps:***  ***- uses: actions/checkout@v2***  ***- name: Set up Python***  ***uses: actions/setup-python@v2***  ***with:***  ***python-version: '3.x'***  ***- name: Install dependencies***  ***run: |***  ***python -m pip install --upgrade pip***  ***pip install unittest***  ***- name: Run tests***  ***run: |***  ***python -m unittest discover tests***  ***- name: Increment version***  ***run: |***  ***version=$(cat version.txt)***  ***new\_version=$(echo $version | awk -F. -v OFS=. '{$NF++; print}')***  ***echo $new\_version > version.txt***  ***git config --global user.name "github-actions"***  ***git config --global user.email "github-actions@github.com"***  ***git add version.txt***  ***git commit -m "Atualiza para versão $new\_version"***  ***git push***  Faça uma nova alteração e veja que o arquivo version.txt será atualizado automaticamente.  **Cenário 4: Atualização Automática de Dependências e Testes de Integração**  Objetivo: Implementar um fluxo CI/CD que atualiza automaticamente as dependências do projeto, executa testes de integração para verificar se a atualização não quebrou o código e realiza commit/push caso os testes sejam bem-sucedidos.  Passos Detalhados:  Configuração Inicial do Projeto:  Crie um repositório no GitHub chamado python-dependency-update.  No VS Code, clone o repositório:  ***git clone https://github.com/seu-usuario/python-dependency-update.git***  ***cd python-dependency-update***  Crie um arquivo requirements.txt para listar as dependências do projeto. Exemplo:  ***requests==2.25.1***  ***pytest==6.2.3***  **Criação de um Script de Atualização de Dependências:**  Crie um arquivo update\_dependencies.py, que será responsável por atualizar as dependências do projeto. Ele pode usar o comando pip para atualizar o requirements.txt e verificar se há novas versões:  ***import os***  ***def update\_dependencies():***  ***os.system("pip install --upgrade -r requirements.txt")***  ***print("Dependências atualizadas com sucesso!")***  ***if \_\_name\_\_ == "\_\_main\_\_":***  ***update\_dependencies()***  Este script vai ser executado automaticamente pelo GitHub Actions para atualizar as dependências do projeto.  **Configuração do GitHub Actions:**  Crie uma pasta .github/workflows e dentro dela, crie um arquivo ci.yml com o seguinte conteúdo:  ***name: CI/CD Pipeline - Dependency Update***  ***on:***  ***push:***  ***branches:***  ***- main***  ***schedule:***  ***- cron: '0 3 \* \* \*' # Executa diariamente às 3 da manhã***  ***jobs:***  ***update-dependencies:***  ***runs-on: ubuntu-latest***  ***steps:***  ***- name: Checkout código***  ***uses: actions/checkout@v3***  ***- name: Configurar Python***  ***uses: actions/setup-python@v4***  ***with:***  ***python-version: '3.x'***  ***- name: Instalar dependências***  ***run: |***  ***python -m pip install --upgrade pip***  ***pip install -r requirements.txt***  ***- name: Atualizar dependências***  ***run: python update\_dependencies.py***  ***- name: Executar testes de integração***  ***run: pytest tests/***  ***- name: Commit e Push (se os testes passarem)***  ***run: |***  ***git config --local user.email "seu-email@exemplo.com"***  ***git config --local user.name "seu-nome"***  ***git add requirements.txt***  ***git commit -m "Atualizar dependências automaticamente"***  ***git push***  ***if: success()***  Esse pipeline será ativado automaticamente toda vez que houver um push para o branch main.  Criação de Testes de Integração:  Crie uma pasta tests/ e dentro dela, crie o arquivo test\_integration.py para garantir que as dependências atualizadas não quebrem o código:  ***import requests***  ***def test\_requests():***  ***response = requests.get('https://api.github.com')***  ***assert response.status\_code == 200***  Este teste simples verifica se a biblioteca requests está funcionando corretamente após a atualização.  Execução Inicial:  Suba o projeto no GitHub:  ***git add .***  ***git commit -m "Setup inicial para atualização automática de dependências"***  ***git push origin main***  O GitHub Actions será ativado automaticamente, verificando se há atualizações nas dependências, rodando os testes de integração e fazendo o push das mudanças.  Simulação de Alterações:  Simule uma atualização nas dependências modificando manualmente o arquivo requirements.txt e subindo as mudanças:  ***requests==2.26.0***  Isso vai ativar novamente o GitHub Actions, que vai verificar se os testes ainda passam após a atualização e, se tudo estiver correto, vai comitar e subir a versão atualizada.  Resultados Esperados:  O sistema atualizará automaticamente as dependências listadas no requirements.txt, rodará os testes de integração para verificar a compatibilidade e, caso tudo esteja correto, fará o commit e push das atualizações no repositório.  **TAREFA 3 – FINALIZAÇÃO**   1. Coloque no fim o nome e RA dos alunos presentes na atividade no cartão de hoje; 2. Coloque o cartão na lista EM VALIDAÇÃO. 3. Mande email para o professor com a URL do projeto no GITHUB:   [flavio.santarelli@pro.fecaf.com.br](mailto:flavio.santarelli@pro.fecaf.com.br)  SUCESSO A TODOS! |