ADS /GTI

SPRINT 1 – MISSÃO 2

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.

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.

MISSÃO 3

VALE 20% DA NOTA AC-1 (0,5)

Etapa 1 - Projeto (GITHUB - método Kanban):

1. No GitHub acessar conta utilizada de forma oficial para interação com o professor;
2. Em BACKLOG criar o item "MISSÃO-3"
3. Colocar "MISSÃO-3" na lista EM EXECUÇÃO;
4. Iniciar o processo de execução de testes;

ETAPA 2 – TESTES UNITÁRIOS INTEGRADOS:

Validando Tarefas com Pytest e GitHub Actions

Objetivo: Criar um validador de tarefas simples, escrever testes unitários para garantir seu funcionamento e automatizar a verificação da qualidade do código a cada alteração usando um fluxo de Integração Contínua (CI).

Cenário: Vamos imaginar que estamos construindo um sistema de gerenciamento de tarefas. Uma parte crucial desse sistema é garantir que uma nova tarefa só possa ser criada se atender a certas regras de negócio. Nossa função será validar um dicionário que representa uma tarefa.

Regras de Negócio para uma Tarefa Válida:

1. Deve ter uma chave "titulo".

2. O valor de "titulo" não pode ser uma string vazia.
3. Deve ter uma chave "prioridade".
4. O valor de "prioridade" deve ser um dos seguintes: "baixa", "media" ou "alta".

Se qualquer uma dessas regras for violada, o validador deve lançar uma exceção (ValueError) com uma mensagem de erro clara.

Passo 0: Preparando o Ambiente e o Código a ser Testado

Primeiro, vamos criar a estrutura do nosso projeto. Peça aos alunos para criarem a seguinte estrutura de pastas e arquivos:

```
projeto_de_testes/
├── src/
│   ├── __init__.py
│   └── task_validator.py
└── tests/
    ├── __init__.py
    └── test_task_validator.py
```

- **src/**: Pasta para o código fonte da nossa aplicação.
- **tests/**: Pasta para nossos códigos de teste.
- **__init__.py**: Arquivos vazios que indicam ao Python que src e tests são pacotes.

1. Crie o Código da Aplicação

Dentro do arquivo src/task_validator.py, coloque o seguinte código Python. Esta é a função que iremos testar.

https://github.com/PROFSANTARELLI/TESTES-QA-2025/tree/main/MISSOES/MISSAO-3/Projeto-teste

Passo 1: Escrever o Teste com Pytest

Agora vamos criar os casos de teste para validar nossa função validate_task. O objetivo é cobrir tanto os cenários de sucesso quanto os de falha.

Abra o arquivo tests/test_task_validator.py e adicione o seguinte código:

https://github.com/PROFSANTARELLI/TESTES-QA-2025/tree/main/MISSOES/MISSAO-3/Projeto-teste

Passo 2: Executar os Testes Localmente

Antes de automatizar, vamos garantir que tudo funciona em nossa máquina.

1. Instale o Pytest
AbrA o terminal, navegar até a pasta raiz `projeto_de_testes/` e instalar o Pytest:

pip install pytest

2. Rode os Testes

Ainda no terminal, na pasta raiz do projeto, execute o seguinte comando:

Pytest

O Pytest irá descobrir e executar automaticamente os testes no diretório `tests/`.

Resultado Esperado:O terminal deve exibir uma saída indicando que todos os 5 testes passaram com sucesso:

Se todos os testes passaram, vamos para o próximo passo!

Agora, vamos configurar um fluxo de trabalho (workflow) para que o GitHub execute nossos testes automaticamente toda vez que enviarmos código novo para o repositório.

Envie o Código para o GitHub:

No terminal, dentro da pasta `projeto_de_testes/`, execute os seguintes comandos para enviar o projeto para o GitHub (substitua a URL pela do repositório deles):

git init
git add .
git commit -m "feat: Adiciona validador de tarefas e testes unitarios"
git branch -M main
git remote add origin https://github.com/SEU-USUARIO/SEU-REPOSITORIO.git git push -u origin main

3. Crie o Workflow do GitHub Actions

1. Na pasta raiz do projeto (projeto_de_testes/), crie uma nova estrutura de diretórios: .github/workflows/.
2. Dentro de workflows, crie um arquivo chamado ci.yml.

```
projeto_de_testes/
├── .github/
│   └── workflows/
│       └── ci.yml
├── src/
│   └── ...
└── tests/
    └── ...
```

1. Abra o arquivo ci.yml e cole o seguinte conteúdo:

https://github.com/PROFSANTARELLI/TESTES-QA-2025/tree/main/MISSOES/MISSAO-3/Projeto-teste

4. Envie o Workflow para o GitHub

Salve o arquivo ci.yml e envie a alteração para o GitHub:

```
git add .
git commit -m "ci: Adiciona workflow do GitHub Actions para testes"
git push
```

5. Verifique o Resultado Final

1. Vá para a página do seu repositório no GitHub.
2. Clique na aba **"Actions"**.
3. Você verá seu workflow ("CI de Testes Python") em execução ou já concluído.
4. Clique no workflow para ver os detalhes. Você verá que um job foi executado para cada versão do Python (3.8, 3.9 e 3.10).
5. Ao clicar em um dos jobs, você pode expandir o passo "Rodar os testes" e ver a mesma saída do Pytest que você viu no seu terminal local, com os "5 passed".
6. Faça alguma modificação no código e rode novamente o processo
7. Avalie os resultados
8. Conserte o código e rode novamente

ETAPA 3 - FINALIZAÇÃO:

5. Após execução e cópia dos Testes no cartão MISSÃO 3, vocês irão pular duas linhas após a o último resultado e inserir nome completo e RA dos alunos presentes;

6. Grave (comittar) as informações e feche o cartão.