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.

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 1 (0,5 ponto)

Início: **15/08** – Término: **05/09**. Vale 0,5 ponto na AC-1 e presenças nas aulas.
Composto por 4 missões que se complementam para a entrega total do projeto:

- Missão 1: Plano do Projeto e Preparação – Vale 10% da AC-1 - CONCLUÍDO
- Missão 2: Testes Funcionais – Vale 30% da AC-1
- Missão 3: Testes Não Funcionais – Vale 30% da AC-1
- Missão Final: Documentação e entrega final – Vale 30% da AC-1

MISSÃO 1

Vale 30% da nota AC-1

TAREFA 1:

1. Compreender o exemplo abaixo sobre o Pytest:

O Pytest é um framework de testes para Python, amplamente utilizado para escrever testes automatizados de software. Oferece funcionalidades poderosas, como fixtures, parametrização e relatórios claros.

2. Abra seu GITHUB para que você execute as tarefas e salve seu modelo e exemplo criado de teste PYTEST

3. Configurando o Pytest: abra o Google Colab, o Jupiter ou o Anaconda, como preferir (professor irá usar o Colab)

4. Primeiro, instale o Pytest no seu ambiente de desenvolvimento:

```
!pip install pytest
```

5. Escrevendo o Código e os Testes: vamos escrever uma função simples e criar testes para ela. A função será uma calculadora básica.

```
# Criando o arquivo de teste dentro do Google Colab
with open("test_calculadora.py", "w") as f:
    f.write("""
import pytest

# Funções que serão testadas
def soma(a, b):
    return a + b

def subtracao(a, b):
    return a - b

def multiplicacao(a, b):
    return a * b

def divisao(a, b):
    if b == 0:
        raise ValueError("Não pode dividir por zero!")
return a / b

# Testes
def test_soma():
    assert soma(3, 2) == 5
    assert soma(-1, 1) == 0

def test_subtracao():
    assert subtracao(5, 3) == 2
    assert subtracao(10, 10) == 0

def test_multiplicacao():
    assert multiplicacao(3, 3) == 9
    assert multiplicacao(4, -2) == -8

def test_divisao():
    assert divisao(10, 2) == 5
    with pytest.raises(ValueError):
```

```
    divisao(10, 0)
"""")
```

6. Executando os Testes;

```
!pytest test_calculadora.py
```

Explicação

Esse método inclui as funções a serem testadas diretamente no arquivo de teste, evitando problemas de importação e simplificando o fluxo no ambiente do Colab. Para casos mais avançados, onde você deseja manter a separação entre código e testes, uma abordagem com módulos externos seria mais adequada, mas para a aula prática no Colab, essa solução resolve o problema de importação.

7. Exemplo de FIXTURE ;

```
import pytest

@pytest.fixture
def dados():
    return {"a": 6, "b": 2}

def test_soma_fixture(dados):
    assert dados["a"] + dados["b"] == 8
```

TAREFA 2:

8. Escreva um código Python para poder utilizar o PYTEST;

9. Teste usando o PyTest no conceito TDD;

10. Print os resultados num documento Word;

11. Suba o código criado no GIT e comitê-o.

12. Anexe o word criado.

FIM – SUCESSO A TODOS!!!