ATIVIDADE PRÁTICA DA DISCIPLINA METODOLOGIAS/ MÉTODOS ÁGEIS

NOME DO CURSO

WERRICSSON MARCONN DA SILVA SANTOS – 4231021
PROFª. MARIANE G B FERNANDES

BAURU

2024

História de Usuário: O empresário Felipe Fernandes precisa realizar a automatização do sistema de sua startup AUTO CENTER FERNANDES. Atualmente o empresário disponibiliza em sua startup produtos automotivos de modo geral. Mas o empresário não tem nenhum software para realizar as seguintes funções: código do produto; marca do produto, quantidade dos produtos em estoque; valor unitário do produto; dados do cliente (nome, CPF, e-mail, contato, endereço e histórico de compras efetuadas e devoluções/trocas); impressão de notas fiscais das compras realizadas pelos clientes; Gastos mensais com funcionários; Gastos mensais básicos (energia e água); entrada/saída de produtos; e os lucros da empresa (mensal e anual). Além disso, Felipe precisará ter neste software dois tipos de login, um administrativo (terão acesso a todos os dados de sua startup e dos clientes) e outro login para seus funcionários (sem o demonstrativo de rendimentos que a startup ganha por dia/mês/ano e gastos gerais da empresa). Seu desafio é pensar como irá desenvolver futuramente um software que atenda a demanda do empresário Felipe para automatizar a startup AUTO CENTER FERNANDES.

A partir da **HISTÓRIA DE USUÁRIO** responda as seguintes perguntas:

1. De acordo com Sutherland e Sutherland (2019, p. 17) “Scrum é uma metodologia ágil para gerenciar projetos complexos, em que não se conhece todas as etapas ou necessidades. Ela se baseia em valores, princípios e práticas que estimulam a colaboração, a criatividade e a adaptação às mudanças.” Posto isto, realize o gerenciamento do Método Scrum para a história de usuário da startup AUTO CENTER FERNANDES **utilizando a ferramenta TRELLO**. No seu gerenciamento você precisará mostrar os seguintes itens: Lista de Backlog; Linha de tempo (com o responsável pela tarefa e citar se tal tarefa está em andamento, realizada, em teste ou em atraso; Citar a quantidade de Sprints dentro de seu gerenciamento e com as devidas descrições). **NÃO SERÁ PERMITIDO ENTREGAR O LINK DO TRELLO.**

R.: O projeto será realizado em 4 sprints, as imagens abaixos mostram etapas da primeira sprint, contendo os requisitos necessários para responder as atividades deste trabalho.

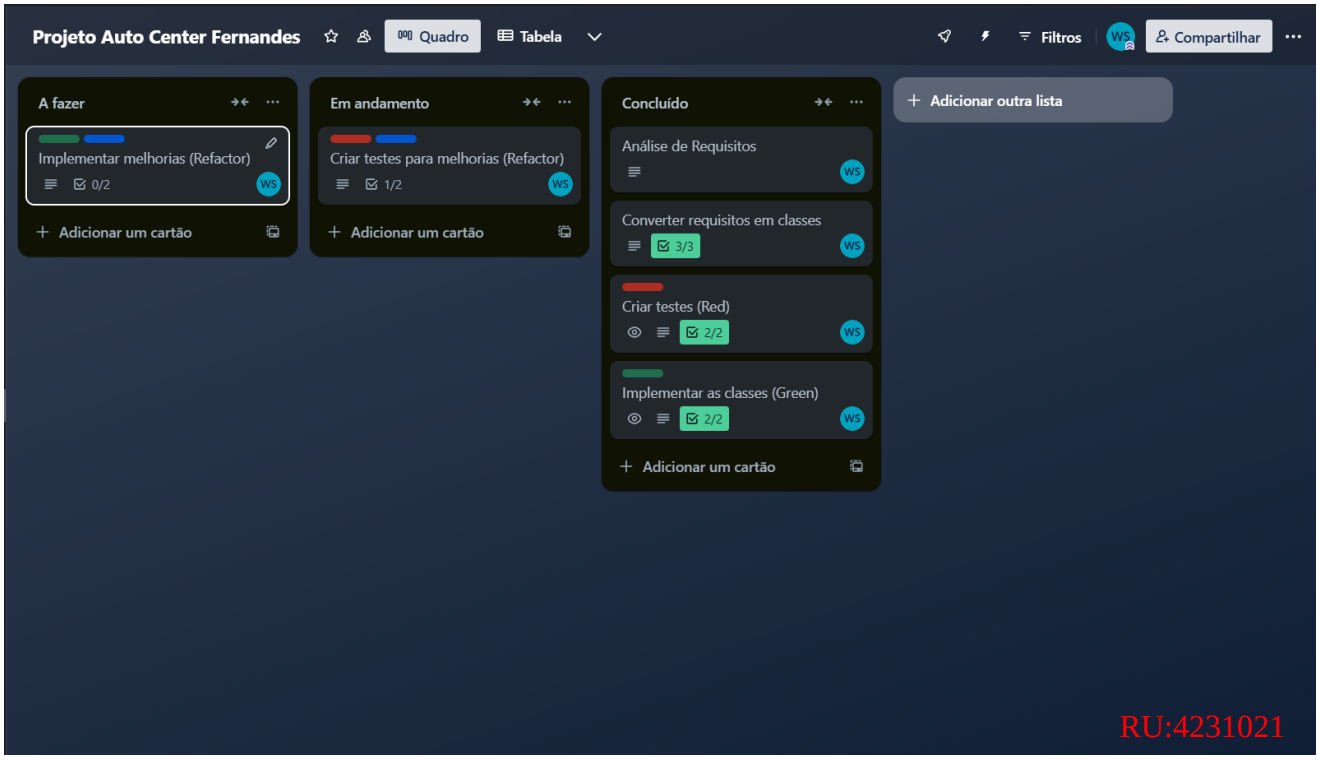

Figura: Quadro TRELLO mostrando as atividades que estão a fazer, em andamento e concluídas (Todos os itens do backlog estão dispostos neste quadro).

Figura: Linha do tempo mostrando a quantidade de cartões em cada etapa por dia.

2. “A conversão de requisitos em classes é uma etapa crucial no desenvolvimento de software orientado a objetos. Nesse processo, os requisitos funcionais e não funcionais identificados são analisados e transformados em classes, que são as unidades fundamentais de estrutura e comportamento do sistema”. Com base nisto, faça a conversão dos resquistos em classes, a partir da história de usuário da startup AUTO CENTER FERNANDES. Apresentar no mínimo 3 classes.

R.:

Classe Funcionário:

Atributos da classe:

- idFuncionario: integer
- nome: string
- cpf: string
- email: string
- contato: string
- endereco: string
- cargo: string
- departamento: string
- salario: float
- admin: boolean

Classe Cliente:

Atributos da classe:

- idCliente: integer
- nome: string
- cpf: string
- email: string

- contato: string
- endereco: string
- comprasEfetuadas: array
- devolucoesETrocas: array

Classe Produto:

Atributos da classe:

- idProduto: integer
- marca: string
- tipo: string
- quantidade: integer
- valor: float

3. Para finalizar o projeto com sucesso, você precisará colocar em prática seus conhecimentos sobre a fase de teste, mais especificamento sobre **o Teste TDD**. Você foi destinado a testar duas classes programadas em Python, uma classe para o cadastro produtos automotivos e outra classe para cadastrar clientes. Além de desenvolver as DUAS classes, você precisará mostrar que as classes irá retornar com sucesso os dados, ou seja, colar o código identado e com comentários. Porm fim, colocar a imagem do terminar sendo executado sem erros com os dados correto das classes testadas.

4.

Dicas para realizar o teste TDD em python:

✓ Escreva testes iniciais: Comece escrevendo testes simples para cada funcionalidade que deseja implementar.

✓ Escreva o código mínimo: Implemente o código mínimo para fazer os testes passarem.

✓ Refatore (se necessário): Após os testes passarem, você pode refatorar o código para torná-lo mais limpo ou eficiente.

✓ Repita: Escreva mais testes para cobrir outros casos e continue iterando até que a funcionalidade seja completa.

R.: Pelo fato de eu ter implementado os métodos e variáveis em português, o código é alto explicativo, por isso não ví necessidade de colocar muitos comentários. Lembrando que eu preferi

dividir em dois arquivos para deixar melhor organizado, você pode verificar os passos (Red, Green, Refactor) a partir dos commits no repositório do projeto, segue link para o repositório: https://github.com/Werricsson-Santos/tdd_uninter_metodologias_ageis

```
import unittest

from classes import Produto, Funcionario

# Iniciamos implementando os testes para depois criar as classes ou features.

class TestProduto(unittest.TestCase):
    def test_criacao_produto(self):
        produto = Produto(1, "Bosch", "Filtro de óleo", 20, 50.00)

        #Exibe as informações do produto antes de realizar os testes
        produto.exibir_informacoes()

        self.assertEqual(produto.idProduto, 1)
        self.assertEqual(produto.marca, "Bosch")
        self.assertEqual(produto.tipo, "Filtro de óleo")
        self.assertEqual(produto.quantidade, 20)
        self.assertEqual(produto.valor, 50.00)

    def test_calculo_valor_total(self):
        produto = Produto(102, "NGK", "Velas de ignição", 4, 35.00)
        self.assertEqual(produto.calcular_valor_total(), 140.00)

    def test_quantidade_insuficiente(self):
        produto = Produto(103, "Valeo", "Pastilha de freio", 5, 120.00)
        self.assertFalse(produto.tem_estoque(10))
        self.assertTrue(produto.tem_estoque(5))

class TestFuncionario(unittest.TestCase):
    def test_criacao_funcionario(self):
        funcionario = Funcionario(1, "Carlos Santos", "12345678900",
"carlos_santos@autocenterfernandes.com", "999999999",
                                  "Rua A, 123", "Mecânico", "Oficina", 3500.00,
False)

        #Exibe as informações do funcionário antes de realizar os testes
        funcionario.exibir_informacoes()

        self.assertEqual(funcionario.idFuncionario, 1)
        self.assertEqual(funcionario.nome, "Carlos Santos")
        self.assertEqual(funcionario.cpf, "12345678900")
        self.assertEqual(funcionario.email, "carlos_santos@autocenterfernandes.com")
        self.assertEqual(funcionario.contato, "999999999")
        self.assertEqual(funcionario.endereco, "Rua A, 123")
```

```
        self.assertEqual(funcionario.cargo, "Mecânico")
        self.assertEqual(funcionario.departamento, "Oficina")
        self.assertEqual(funcionario.salario, 3500.00)
        self.assertFalse(funcionario.admin)


    def test_acesso_admin(self):
        mecanico = Funcionario(1, "Carlos Santos", "12345678900",
"carlos_santos@autocenterfernandes.com", "999999999",
                               "Rua A, 123", "Mecânico", "Oficina", 3500.00,
False)
        self.assertFalse(mecanico.admin)
        self.assertEqual(mecanico.acessar_area_admin(), "Você não possui acesso
administrativo.")

        gerente_admin = Funcionario(3, "José Marinho", "11223344556",
"jose_marinho@autocenterfernandes.com", "777777777",
                                "Rua C, 789", "Gerente", "Administração", 7000.00,
True)
        self.assertTrue(gerente_admin.admin)
        self.assertEqual(gerente_admin.acessar_area_admin(), "Acesso liberado.")


if __name__ == "__main__":
    unittest.main()
```

Figura: Código que implementa os testes das classes, arquivo testes.py.

```
#Após a criação dos testes, criamos as classes ou features a fim de atingir o
objetivo de passar em todos os testes.

#Implementação da classe Produto
class Produto:
    def __init__(self, idProduto, marca, tipo, quantidade, valor):
        self.idProduto = idProduto
        self.marca = marca
        self.tipo = tipo
        self.quantidade = quantidade
        self.valor = valor

    def exibir_informacoes(self):
        print("\n")
        print("Tipo: ", self.tipo)
        print("Marca: ", self.marca)
        print("Quantidade: ", self.quantidade)
        print("Valor: ", self.valor)
        print("\n")

    #Calcula o valor total de acordo com a quantidade
    def calcular_valor_total(self):
        return self.quantidade * self.valor
```

```
    #Verifica se há quantidade suficiente no estoque a partir da solicitação do
cliente.
    def tem_estoque(self, quantidade_requerida):
        return self.quantidade >= quantidade_requerida

#Implementação da classe funcionário
class Funcionario:
    def __init__(self, idFuncionario, nome, cpf, email, contato, endereco, cargo,
departamento, salario, admin):
        self.idFuncionario = idFuncionario
        self.nome = nome
        self.cpf = cpf
        self.email = email
        self.contato = contato
        self.endereco = endereco
        self.cargo = cargo
        self.departamento = departamento
        self.salario = salario
        self.admin = admin

    def exibir_informacoes(self):
        print("\n")
        print("Nome: ", self.nome)
        print("Cargo: ", self.cargo)
        print("Email: ", self.email)
        print("Contato: ", self.contato)
        print("Endereço: ", self.endereco)
        print("\n")

    #Simula acesso a area administrativa
    def acessar_area_admin(self):
        #O acesso só é liberado se o funcionário for admin.
        response = "Acesso liberado." if self.admin else "Você não possui acesso
administrativo."

        return response
```

Figura: Código que implementa as classes, arquivo classes.py.

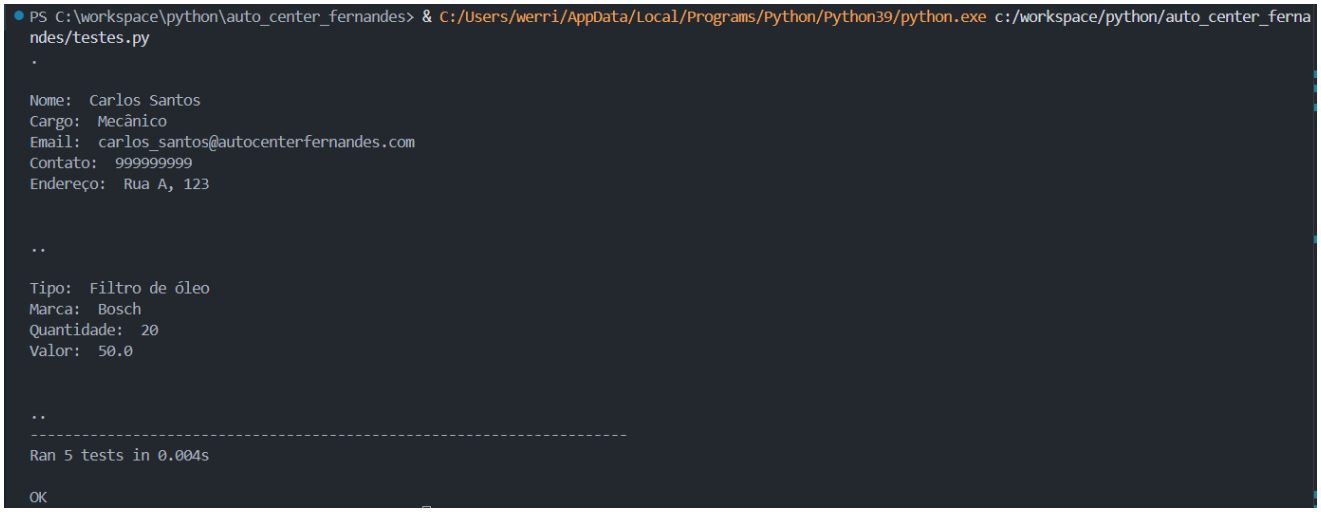

Figura: Print do terminal após a execução do arquivo testes.py.