

Roteiro do Documento e Processo de Testes

1. Introdução - Importância dos testes unitários:

Os testes de software têm como principal objetivo garantir que o sistema funcione corretamente e evitar falhas durante o uso real. Eles são parte essencial do processo de desenvolvimento, pois permitem identificar erros, confirmar se as correções surtiram efeito e assegurar que novas alterações não prejudiquem o funcionamento geral do sistema.

No caso do nosso projeto, que parte de um código legado com erros, os testes são ainda mais importantes. Eles nos ajudarão a entender onde estão os problemas, acompanhar o progresso das correções e garantir que as funcionalidades continuem operando após cada mudança no código.

2. Planejamento - Calendário de Testes

Etapa (datas)	Objetivo	Descrição
Semana 01	Testes iniciais	Executar o código original e registrar os erros encontrados.
Semana 02	Análise e correção	Corrigir os erros detectados, mantendo a funcionalidade básica.
Semana 03	Testes de verificação	Reexecutar os testes para confirmar se as correções funcionaram.

Semana 04	Refatoração	Melhorar a estrutura do código (nomes, organização, legibilidade).
Semana 05	Testes finais	Executar todos os testes novamente para confirmar a estabilidade do sistema.

3. Tipos de Testes Realizados

Para esse projeto, utilizaremos testes unitários, que são testes aplicados em funções individuais do código. Eles ajudam a validar partes específicas do sistema sem precisar executar o programa todo.

Função testada	Nome do teste	O que o teste verifica	Resultado esperado
adicionar_usuario	test_adicionar_usuario_valido	Verifica se um nome válido é adicionado corretamente à lista de usuários.	O nome deve aparecer na lista após a adição.
adicionar_usuario	test_adicionar_usuario_vazio	Garante que nomes vazios não sejam aceitos.	Nenhum usuário deve ser adicionado.
adicionar_usuario	test_adicionar_usuario_duplicado	Testa se o sistema impede a adição de nomes repetidos.	Cada nome deve aparecer apenas uma vez na lista.
remover_usuario	test_remover_usuario_existente	Verifica se o sistema remove corretamente um usuário existente.	O nome removido não deve mais aparecer na lista.
remover_usuario	test_remover_usuario_inexistente	Testa o comportamento	O sistema deve lidar

	existente	ao tentar remover um usuário que não existe.	com o erro sem interromper a execução.
buscar_usuario	test_buscar_usuario_existe	Verifica se o sistema encontra corretamente o usuário buscado.	Deve retornar o nome do usuário procurado.
buscar_usuario	test_buscar_usuario_inexistente	Garante que o sistema retorne "None" se o usuário não for encontrado.	Retorno deve ser None, sem erro.
listar_usuarios	test_listar_usuarios	Confirma se a listagem retorna todos os usuários corretamente.	Deve retornar a lista completa de usuários cadastrados.
listar_usuarios	test_listar_usuarios_vazio	Verifica se o sistema trata corretamente uma lista vazia.	Deve retornar uma lista vazia ([]).

Esses testes serão desenvolvidos com a biblioteca unittest, que já vem embutida no Python.

4. Execução dos Testes - Ciclo de Correção

Durante o desenvolvimento, seguiremos o seguinte processo:

Rodar os testes iniciais	detectar onde o código falha.
Registrar os resultados	anotar quais testes falharam e por quê.
Corrigir o código	ajustar as funções com erro.
Rodar novamente os testes	confirmar se a correção funcionou.
Refatorar o código	melhorar a clareza e a legibilidade sem alterar o funcionamento.

Executar os testes finais	garantir que todas as funções continuam operando corretamente.
---------------------------	--

Esse processo será repetido até que todos os testes passem e o sistema esteja funcional e bem estruturado.

5. Explicação do código teste

criar um arquivo com o teste (`test_sistema.py`) e um arquivo para o código fonte (`sistema.py`), ambos dentro da mesma pasta

`test_sistema.py`

```
import unittest # módulo para realização de testes no Python
from sistema_legado import adicionar_usuario, remover_usuario,
buscar_usuario, listar_usuarios

from test_helpers import (
    adicionar_usuario,
    remover_usuario,
    buscar_usuario,
    listar_usuarios
)

class TestSistema(unittest.TestCase):

    # -----
    # TESTES DA FUNÇÃO ADICIONAR
    # -----


    def test_adicionar_usuario_valido(self):
        """Deve adicionar um nome corretamente na lista."""
        usuarios = []
        adicionar_usuario(usuarios, "Maria")
        self.assertIn("Maria", usuarios)

    def test_adicionar_usuario_vazio(self):
```

```
"""Não deve adicionar usuário com nome vazio."""
usuarios = []
adicionar_usuario(usuarios, "")
self.assertEqual(len(usuarios), 0)

def test_adicionar_usuario_duplicado(self):
    """Não deve permitir duplicatas."""
    usuarios = ["Ana"]
    adicionar_usuario(usuarios, "Ana")
    self.assertEqual(usuarios.count("Ana"), 1)

# -----
# TESTES DA FUNÇÃO REMOVER
# -----

def test_remover_usuario_existente(self):
    """Deve remover o usuário corretamente."""
    usuarios = ["João", "Maria"]
    remover_usuario(usuarios, "Maria")
    self.assertNotIn("Maria", usuarios)

def test_remover_usuario_inexistente(self):
    """Não deve quebrar se tentar remover alguém que não existe."""
    usuarios = ["Carlos"]
    try:
        remover_usuario(usuarios, "Zeca")
        resultado = True
    except Exception:
        resultado = False
    self.assertTrue(resultado)

# -----
# TESTES DA FUNÇÃO BUSCAR
# -----

def test_buscar_usuario_existente(self):
    """Deve encontrar o usuário correto."""
    usuarios = ["Ana", "Bruno"]
    resultado = buscar_usuario(usuarios, "Ana")
    self.assertEqual(resultado, "Ana")

def test_buscar_usuario_inexistente(self):
    """Deve retornar None quando o usuário não for encontrado."""

```

```
usuarios = ["Ana"]
resultado = buscar_usuario(usuarios, "Pedro")
self.assertIsNone(resultado)

# -----
# TESTES DA FUNÇÃO LISTAR
# -----


def test_listar_usuarios(self):
    """Deve retornar todos os usuários corretamente."""
    usuarios = ["Ana", "Bruno", "Carla"]
    resultado = listar_usuarios(usuarios)
    self.assertEqual(resultado, ["Ana", "Bruno", "Carla"])

def test_listar_usuarios_vazio(self):
    """Deve retornar lista vazia se não houver usuários."""
    usuarios = []
    resultado = listar_usuarios(usuarios)
    self.assertEqual(resultado, [])

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

6. Evidências e Resultados

Teste Inicial:

ERRO:

```
MINGW64:/c/Users/DreEm/OneDrive/Documentos/A3 - Calvetti
DreEm@LAPTOP-OLDG81T1 MINGW64 ~/OneDrive/Documentos/A3 - Calvetti
$ ls
sistema_legado.py test_helpers.py test_sistema.py test_validar.py
DreEm@LAPTOP-OLDG81T1 MINGW64 ~/OneDrive/Documentos/A3 - Calvetti
$ python -m unittest test_sistema
EEEEEEEF

ERROR: test_adicionar_usuario_duplicado (test_sistema.TestSistema.test_adicionar_usuario_duplicado)
Não deve permitir duplicatas.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 32, in test_adicionar_usuario_duplicado
    adicionar_usuario(usuarios, "Ana")
TypeError: adicionar_usuario() takes 1 positional argument but 2 were given

ERROR: test_adicionar_usuario_valido (test_sistema.TestSistema.test_adicionar_usuario_valido)
Deve adicionar um nome corretamente na lista.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 20, in test_adicionar_usuario_valido
    adicionar_usuario(usuarios, "Maria")
TypeError: adicionar_usuario() takes 1 positional argument but 2 were given

ERROR: test_adicionar_usuario_vazio (test_sistema.TestSistema.test_adicionar_usuario_vazio)
Não deve adicionar usuário com nome vazio.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 26, in test_adicionar_usuario_vazio
    adicionar_usuario(usuarios, "")
TypeError: adicionar_usuario() takes 1 positional argument but 2 were given

ERROR: test_buscar_usuario_existente (test_sistema.TestSistema.test_buscar_usuario_existente)
Deve encontrar o usuário correto.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 62, in test_buscar_usuario_existente
    resultado = buscar_usuario(usuarios, "Ana")
               ^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: buscar_usuario() takes 1 positional argument but 2 were given

ERROR: test_buscar_usuario_inexistente (test_sistema.TestSistema.test_buscar_usuario_inexistente)
Deve retornar None quando o usuário não for encontrado.
-----
```

```
MINGW64:/c/Users/DreEm/OneDrive/Documentos/A3 - Calvetti
ERROR: test_buscar_usuario_inexistente (test_sistema.TestSistema.test_buscar_usuario_inexistente)
Deve retornar None quando o usuário não for encontrado.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 68, in test_buscar_usuario_inexistente
    resultado = buscar_usuario(usuarios, "Pedro")
               ^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: buscar_usuario() takes 1 positional argument but 2 were given

ERROR: test_listar_usuarios (test_sistema.TestSistema.test_listar_usuarios)
Deve retornar todos os usuários corretamente.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 78, in test_listar_usuarios
    resultado = listar_usuarios(usuarios)
               ^^^^^^^^^^^^^^^^^^^^^^
TypeError: listar_usuarios() takes 0 positional arguments but 1 was given

ERROR: test_listar_usuarios_vazio (test_sistema.TestSistema.test_listar_usuarios_vazio)
Deve retornar lista vazia se não houver usuários.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 84, in test_listar_usuarios_vazio
    resultado = listar_usuarios(usuarios)
               ^^^^^^^^^^^^^^^^^^^^^^
TypeError: listar_usuarios() takes 0 positional arguments but 1 was given

ERROR: test_remover_usuario_existente (test_sistema.TestSistema.test_remover_usuario_existente)
Deve remover o usuário corretamente.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 42, in test_remover_usuario_existente
    remover_usuario(usuarios, "Maria")
               ^^^^^^^^^^
TypeError: remover_usuario() takes 1 positional argument but 2 were given

FAIL: test_remover_usuario_inexistente (test_sistema.TestSistema.test_remover_usuario_inexistente)
Não deve quebrar se tentar remover alguém que não existe.
Traceback (most recent call last):
  File "C:\Users\ DreEm\OneDrive\Documentos\A3 - Calvetti\test_sistema.py", line 53, in test_remover_usuario_inexistente
    self.assertTrue(resultado)
AssertionError: False is not true
-----
```

Nenhuma das funções passou nos testes executados. Foi adicionado um arquivo de helpers, para apoiar o fato do nosso código legado não ter funções básicas para a realização de um teste.

Teste concluído com sucesso: OK

```
DreEm@LAPTOP-OLDG81T1 MINGW64 ~/OneDrive/Documentos/A3 - Calvetti
$ python -m unittest test_sistema
.....
-----
Ran 9 tests in 0.001s
OK
DreEm@LAPTOP-OLDG81T1 MINGW64 ~/OneDrive/Documentos/A3 - Calvetti
$ |
```

Foi utilizado um código minimamente OK, para que passasse nos testes.

7. Conclusão