# Guia de Testes de Integração com `unittest`

## 1. Introdução ao `unittest`

`unittest` é um framework de testes embutido no Python que permite a criação de testes robustos para aplicações. É especialmente útil para garantir a funcionalidade e a integridade de sistemas complexos.

## 2. Configuração do Ambiente de Teste

### Instalação

Para começar a escrever e executar testes, você precisa configurar seu ambiente. Use o arquivo `requirements.txt` para instalar todas as dependências necessárias:

```bash

pip install -r requirements.txt

Estrutura de Diretórios

Organize seus testes em uma pasta tests na raiz do seu projeto. Cada arquivo dentro desta pasta deve conter testes relacionados a uma parte específica da aplicação:

bash

Copiar código

/Tecsus-ETL

/data

/src

/tests

test\_api.py

test\_data\_processing.py

3. Abordagem de Desenvolvimento Bottom-Up

Na nossa equipe, adotamos a técnica de desenvolvimento bottom-up para o backend. Isso envolve começar pelo desenvolvimento e testes das partes mais básicas do sistema (como modelos e funções de acesso a dados) e gradualmente integrar essas partes até formar componentes de nível superior.

4. Escrevendo Testes com unittest

Definição de Casos de Teste

Cada conjunto de testes deve ser definido em uma classe que herda de unittest.TestCase. Por exemplo, um teste básico para o endpoint principal:

import unittest

from fastapi.testclient import TestClient

from myapp.main import app

class TestAPI(unittest.TestCase):

def setUp(self):

self.client = TestClient(app)

def test\_home(self):

response = self.client.get("/")

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json(), {"message": "Hello World"})

Testando Manipulação de Dados com pandas

def test\_data\_processing(self):

from myapp.data\_processing import process\_data

result = process\_data("path/to/test/data.csv")

self.assertTrue(result.empty, "Dataframe should not be empty")

Testando Interações com o Banco de Dados usando SQLAlchemy

def test\_database\_interaction(self):

from myapp.database import get\_user

user = get\_user(user\_id=1)

self.assertIsNotNone(user, "User should not be None")

Asserts

Utilize assertivas para verificar se as condições do teste são atendidas:

self.assertEqual(a, b): Verifica se a é igual a b.

self.assertTrue(x): Verifica se x é verdadeiro.

self.assertFalse(x): Verifica se x é falso.

5. Executando Testes

Use o seguinte comando para executar todos os testes:

python -m unittest discover -s tests

6. Práticas Recomendadas

Isolamento de Teste: Garanta que os testes não dependam uns dos outros.

Cobertura de Teste: Tente cobrir tanto os casos típicos quanto os atípicos.

Documentação: Comente seus testes para explicar o que cada um verifica.

7. Recursos Adicionais

Documentação oficial do unittest

Documentação do FastAPI

Documentação do Pandas

Documentação do SqlAchemy

Este guia abrangente cobre não só o básico dos testes de API, mas também como testar manipulações de dados complexas e interações com o banco de dados, refletindo melhor a complexidade do seu projeto `Tecsus-ETL`.