Execução Avançada e Relatórios

Introdução

Este módulo aborda recursos para otimizar e personalizar a execução dos testes:

- **Execução paralela** para ganhar velocidade.
- **Relatórios HTML** para facilitar a análise dos resultados.
- **Argumentos de linha de comando (CLI)** do Pytest com Playwright para configurar a execução sem alterar código.

---

1 – Execução Paralela

Conceito

Quando o número de testes cresce, a execução pode se tornar lenta.

A execução **paralela** permite que vários testes rodem ao mesmo tempo, aproveitando melhor o hardware disponível.

Explicação

No Pytest, a execução paralela é feita com o plugin **pytest-xdist**.

O parâmetro `-n` define o número de processos simultâneos.

Instalação

```
pip install pytest-xdist
```

Exemplo – Execução paralela

```
pytest -n 2
```

```
pytest -n auto
```

```
# test_validar_acesso_as_paginas.py

from pages.base_page import BasePage
from playwright.sync_api import expect

def test_validar_home(page):
    pagina = BasePage(page)
    pagina.acessar_home()
    expect(page.get_by_role("heading", name="AutomationExercise")).to_be_visible()


def test_validar_produtos(page):
    pagina = BasePage(page)
    pagina.acessar_produtos()
```

```
    expect(page.get_by_role("img", name="Website for practice")).to_be_visible()


def test_validar_carrinho(page):
    pagina = BasePage(page)
    pagina.acessar_carrinho()
    expect(page.get_by_text("Home Shopping Cart")).to_be_visible()


def test_validar_login(page):
    pagina = BasePage(page)
    pagina.acessar_cadastro_login()
    expect(page.get_by_role("heading", name="Login to your account")).to_be_visible()
```

Referência oficial:

🔗 https://pytest-xdist.readthedocs.io/en/stable/

---

2 – Relatórios HTML

Conceito

Os **relatórios HTML** do Pytest ajudam a visualizar rapidamente:

- Quais testes **passaram**
- Quais **falharam**
- Quais foram **ignorados**

Eles são muito usados para **compartilhar resultados** com o time ou para manter um **histórico visual** das execuções.

Explicação

O **pytest-html** é um plugin que gera um arquivo `.html` com todas as informações da execução.

Ele permite:

- Adicionar **metadados** (ex.: data, autor, ambiente)
- **Customizar** o relatório (inserir imagens, textos, links)

Instalação

```
pip install pytest-html
```

Exemplo básico – Gerar relatório HTML

```
pytest --html=report.html
```

Relatório HTML auto-contido

```
pytest --html=report.html --self-contained-html
```

O parâmetro `--self-contained-html` embute CSS e imagens diretamente no arquivo HTML, facilitando o envio por e-mail ou upload, sem depender de arquivos externos.

Referência oficial:

🔗 https://pytest-html.readthedocs.io/en/latest/

Customizando com imagens de erro (Playwright)

Para inserir screenshots automaticamente no relatório **apenas quando um teste falha**, podemos usar o **gancho** (`hook`) `pytest_runtest_makereport`.

📌 O que é o `pytest_runtest_makereport`?

É um **gancho do Pytest** que é executado **após cada fase** de execução de um teste (`setup`, `call`, `teardown`).

Ele retorna um **objeto de relatório** com informações como:

- `when` → fase (`setup`, `call` ou `teardown`)
- `outcome` → resultado (`passed`, `failed`, `skipped`)
- `longrepr` → mensagem de erro, se houver

🔗 **Documentação oficial:**

https://docs.pytest.org/en/stable/reference/reference.html#pytest.hookspec.pytest_runtest_makereport

---

Exemplo no `conftest.py` com Playwright

```
import pytest
import pytest_html
from slugify import slugify  # pip install python-slugify

@pytest.hookimpl(hookwrapper=True)
def pytest_runtest_makereport(item, call):
    """
    Este hook é executado após cada fase do teste.
    Usamos ele para verificar falhas e anexar capturas de tela ao relatório HTML.
    """
    outcome = yield
    report = outcome.get_result()

    # Lista de extras para o relatório HTML
    extras = getattr(report, 'extra', [])

    # Só captura na fase 'call' (execução do teste em si)
    if report.when == 'call':
        xfail = hasattr(report, 'wasxfail')

        try:
            # Captura quando falha (mas não é um XFAIL esperado)
```

```
        if (report.skipped and xfail) or (report.failed and not xfail):
            # Garante nome de arquivo seguro
            screen_file = f"imagens/{slugify(item.nodeid)}.png"

            # page deve estar disponível no teste (fixture do Playwright)
            page = item.funcargs.get("page")
            if page:
                page.screenshot(path=screen_file, full_page=True)
                extras.append(pytest_html.extras.png(screen_file))
    except Exception as e:
        print(f"Erro ao capturar imagem: {e}")

    report.extra = extras
```

Observações

1. `slugify(item.nodeid)`

   Garante que o nome do arquivo não tenha caracteres inválidos.

2. `page = item.funcargs.get("page")`

   Acessa a fixture `page` do Playwright, sem precisar importar nada extra no hook.

3. `pytest_html.extras.png`

   Converte a imagem para o formato aceito pelo relatório HTML.

Exemplo de Execução

```
pytest --html=report.html --self-contained-html
```

4 – Utilizando Marks nos Testes

Explicação

As marcas (`marks`) permitem organizar e filtrar a execução de testes.

Exemplo – Uso de marca no teste

```
import pytest

@pytest.mark.login
def test_login_valido(page):
    page.goto("https://automationexercise.com/login")
    page.fill("input[data-qa='login-email']", "usuario@teste.com")
    page.fill("input[data-qa='login-password']", "senha123")
    page.click("button[data-qa='login-button']")
    assert page.is_visible("text='Logged in as'")
```

Exemplo – Executando apenas testes marcados

```
pytest -m login
```

3 – Argumentos CLI do Playwright com Pytest

Conceito

O Playwright adiciona parâmetros próprios à CLI do Pytest para controlar navegador, device, vídeo, screenshots e tracing.

Explicação

Esses argumentos afetam os fixtures padrão (`browser` , `context` , `page`) e não são aplicados automaticamente quando a fixture é criada manualmente.

Exemplos – Argumentos úteis

```
pytest --browser=firefox
pytest --browser-channel=chrome
pytest --slowmo 1000
pytest --device "iPhone 12"
pytest --tracing=on
pytest --screenshot=only-on-failure
```

Referência oficial:

🔗 https://playwright.dev/python/docs/test-runners#cli-arguments

Boas Práticas

1. Combine **execução paralela** com **relatórios HTML** para obter velocidade e rastreabilidade.
2. Utilize os argumentos CLI do Playwright para mudar configurações sem alterar código.
3. Centralize configurações no `pytest.ini` para padronizar a execução entre toda a equipe.