



TESTES AUTOMATIZADOS EM SALESFORCE:

Uma visão prática de como automatizar testes com Python, Selenium e Pytest

Autor:
RAFAEL CARVALHO FRANÇA

Orientadora:
ZADY CASTAÑEDA SALAZAR

INTRODUÇÃO

- O que é teste de software?
- Qual o papel dos testes?
- Teste manual ou automatizado?

OBJETIVOS

GERAL:

Desenvolver uma automação de testes em ambiente web utilizando Python, Pytest e o Selenium WebDriver, com o objetivo de otimizar processos repetitivos e manuais, garantindo estabilidade e confiabilidade na execução de interações automatizadas em aplicações web.

OBJETIVOS

ESPECÍFICOS:

- Desenvolver casos de teste cobrindo cenários comuns (preenchimento de formulários, navegação e validação de elementos)
- Implementar o padrão POM para organizar e reutilizar elementos de página;
- Utilizar o framework pytest para estruturar, organizar e gerar relatórios dos testes automatizados, explorando recursos como marcadores, fixtures e do BDD (do inglês, Behavior Driven Development);
- Executar testes automatizados registrando capturas de tela como um auxílio visual para validar a correta execução das interações automatizadas;
- Analisar os resultados dos testes;
- Apresentar os resultados do trabalho de forma clara e organizada, destacando os benefícios, além de sugestões para aprimoramentos futuros.

JUSTIFICATIVA

1. Relevância da automação de testes
2. Benefícios da automação
3. Escolha do Selenium WebDriver
4. Salesforce como contexto de aplicação

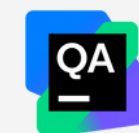
FUNDAMENTAÇÃO TEÓRICA

- Teste de software
- Teste funcional
- Testes Automatizados
- Selenium WebDriver
- Python
- Salesforce
- POM (Page Object Model)
- Aqua IDE



FUNDAMENTAÇÃO TEÓRICA

- Behavior-Driven Development (BDD)
- Pytest
 - Pytest.fixtures
 - Pytest-html
 - Pytest.marks
 - Pytest-bdd



METODOLOGIA

1. Levantamento bibliográfico;

METODOLOGIA

2. Configuração do ambiente de desenvolvimento;

Ferramentas utilizadas:

Ferramenta	Categoria	Versão	Descrição
Google Chrome	Instalado via .exe	130.0.6723.92	Navegador web usado para automação de testes de interface gráfica com Selenium.
Aqua IDE	Instalado via .exe	2024.2.1	Ambiente de desenvolvimento para Python, com suporte a análise e depuração.
Python	Instalado via .exe	3.11.2004	Linguagem de programação versátil, usada para desenvolvimento de aplicações, análise de dados, etc.
Selenium	Biblioteca python instalada via pip	4.25.0	Automação de navegadores para testes, preenchimento de formulários e outras interações online.
pytest-html	Biblioteca python instalada via pip	4.1.2001	Gera relatórios em HTML para testes executados com pytest.
pytest-bdd	Biblioteca python instalada via pip	7.3.2000	Integra pytest com o BDD (Behavior-Driven Development) para testes orientados a comportamento.


METODOLOGIA

2. Configuração do ambiente de desenvolvimento;


Organização das pastas e arquivos:

Pastas	Descrição
Raiz do projeto	Contém arquivos de configuração do pytest (conftest.py), Python (pyproject.toml), .env para as variáveis sensíveis e requirements.txt para instalação das dependências.
/pages	Centraliza as funções de teste por página, incluindo uma página base com funções principais do Selenium.
/tests/data	Armazena arquivos .json usados para validação de labels e preenchimento de campos.
/tests/features	Contém arquivos .feature escritos em Gherkin, definindo cenários de teste com as etapas a serem executadas.
/tests/step-defs	Armazena os arquivos de definição dos passos de teste, mapeando funções das páginas com os cenários definidos.
/utils/reports	Guarda o último relatório gerado após a execução dos testes, tanto por suite quanto individualmente.
/utils/screenshots	Armazena evidências em .png (capturas de tela) feitas durante a execução dos testes.


▼

 pages


▼

 tests


>

 data


>

 features


>

 step-defs


▼


 utils


▼


 reports


>

 screenshots

 .env

 conftest.py

 pyproject.toml

 requirements.txt

METODOLOGIA

3. Desenvolvimento de casos de teste

3.1 Utilização do framework de testes pytest

conftest.py

```
@pytest.fixture
def driver():
    opt = Options()
    opt.add_experimental_option(
        "prefs", {"profile.default_content_setting_values.notifications": 2}
    )
    driver = webdriver.Chrome(options=opt)
    driver.implicitly_wait(10)
    driver.maximize_window()
    driver.get("https://login.salesforce.com")


    yield driver

    driver.quit()
```

METODOLOGIA

3. Desenvolvimento de casos de teste

3.2 Criação das especificações dos testes usando o BDD



tests/features/ct03_create_account.feature

```
@create_account
Feature: Criar conta no formulário de cadastro

  @successful
  Scenario: Criar uma conta com sucesso
    Given que estou na página de cadastro de conta
    When eu acesso o formulário de criação de conta
    And preencho todos os campos disponíveis
    And eu salvo a conta
    Then a conta deve ser criada com sucesso
    And todos os campos devem estar presentes nos detalhes da conta
```

METODOLOGIA

3. Desenvolvimento de casos de teste

3.3 Implementação do POM



pages/account_page.py

```
account_page = (By.XPATH, '//span[@class="slds-var-p-right_x-small"]')
```



pages/account_page.py

```
def verificar_pagina_cadastro(self):  
    url = os.getenv("ACCOUNT_PAGE_URL")  
    self.access_link(url)  
    self.text_exists_on_screen(AccountLocators.account_page, "Contas")
```



pages/base_page.py

```
class BasePage:  
    def __init__(self, driver):  
        self.driver = driver  
  
    def access_link(self, tab_link):  
        self.driver.get(tab_link)  
  
    def text_exists_on_screen(self, locator, text, timeout=10):  
        try:  
            self.presence_wait(locator, timeout)  
            element = self.find_element(locator)  
  
            assert text in element.text, (  
                f"O texto '{text}' não foi encontrado "  
                f"no elemento com o seletor {locator}!"  
            )  
  
        except TimeoutException:  
            pytest.fail(  
                f"O seletor {locator} não foi encontrado na tela!",  
                pytrace=True,  
            )
```

METODOLOGIA

4. Criação, execução e validação dos testes

```
tests/step-defs/test_ct03_create_account.py

@pytest.fixture
def ap(driver):
    return AccountPage(driver, screenshot=True)

@pytest.mark.account
@scenario("../features/ct03_create_account.feature", "Criar uma conta com sucesso")
def test_ct03_create_account():
    pass

@given("que estou na página de cadastro de conta")
def acessar_pagina_cadastro(driver, ap):
    login_page = LoginPage(driver)
    login_page.login_com_sucesso()
    ap.verificar_pagina_cadastro()

@when("eu acesso o formulário de criação de conta")
def acessar_formulario_criacao_conta(ap):
    ap.abrir_formulario_criacao_conta()
```

```
@when("preencho todos os campos disponíveis")
def preencher_campos_disponiveis(ap, load_account_data):
    ap.preencher_formulario_conta(load_account_data)

@when("eu salvo a conta")
def clicar_criar_conta(ap):
    ap.salvar_conta()

@then("a conta deve ser criada com sucesso")
def verificar_edicao_conta(ap, load_new_account_data):
    ap.verificar_sucesso(
        n_campos=len(load_new_account_data), campos_adicionais=1, campos_agrupados=0
    )


@then("todos os campos devem estar presentes nos detalhes da conta")
def verificar_campos_detalhes(ap, load_account_data):
    ap.verificar_campos_detalhes(load_account_data)
```

RESULTADOS

Comando para execução de todos os testes

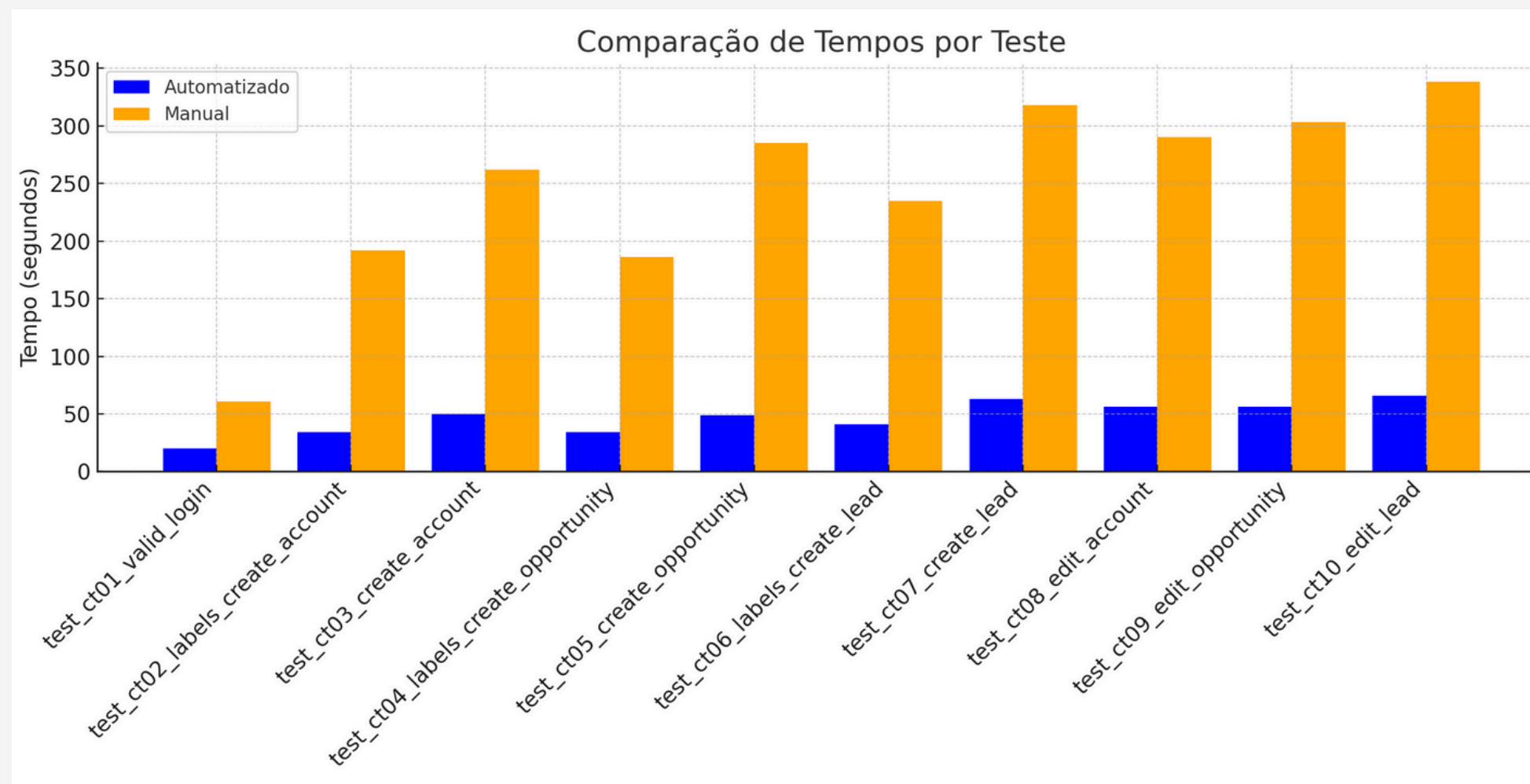
```
pytest -s -vv --html=utils/reports/report.html
```

Arquivo de resultados em HTML

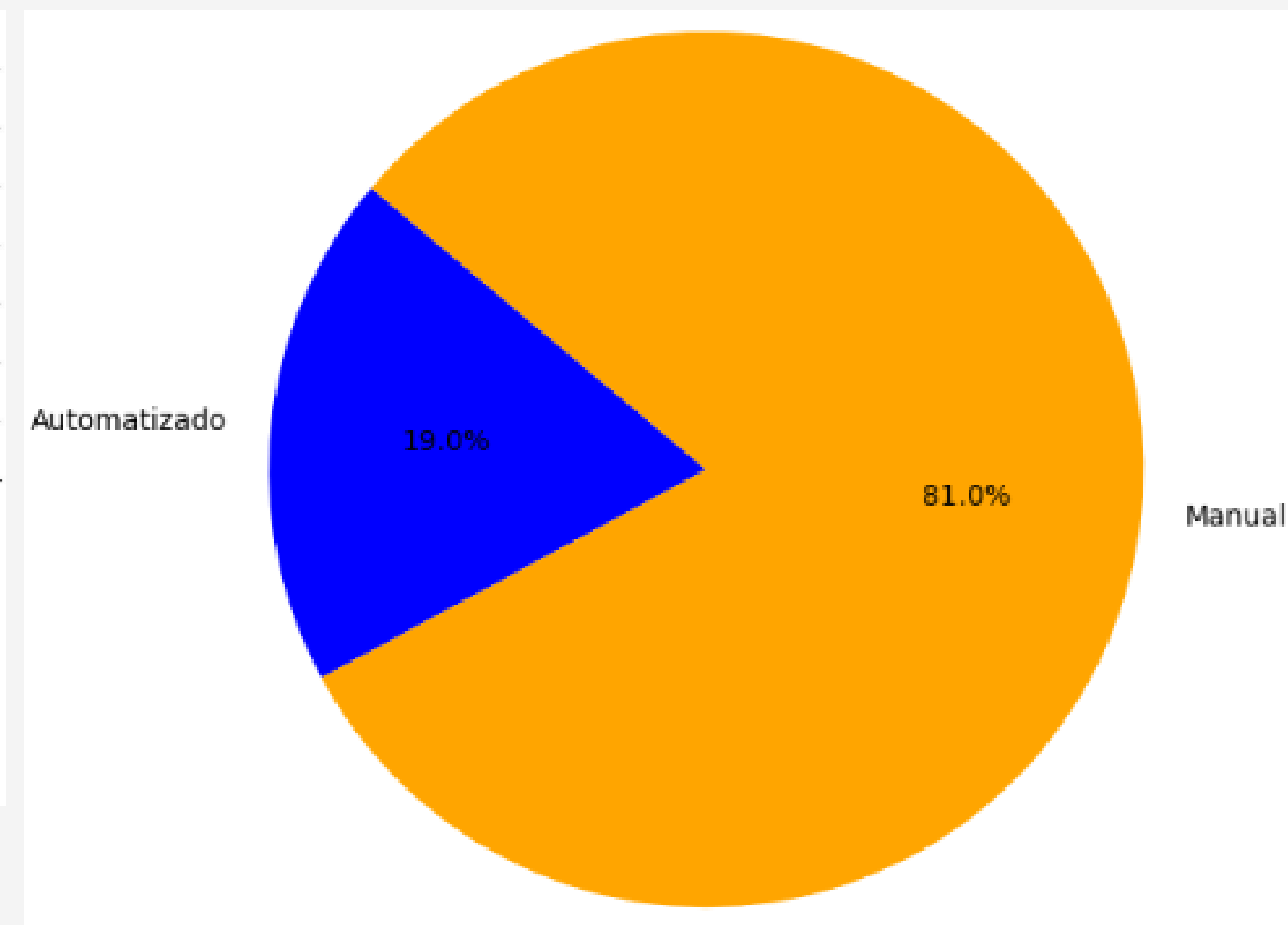
Summary			
10 tests took 00:07:50.			
(Un)check the boxes to filter the results.			
<input checked="" type="checkbox"/> 0 Failed, <input checked="" type="checkbox"/> 10 Passed, <input checked="" type="checkbox"/> 0 Skipped, <input checked="" type="checkbox"/> 0 Expected failures, <input checked="" type="checkbox"/> 0 Unexpected passes, <input checked="" type="checkbox"/> 0 Errors, <input checked="" type="checkbox"/> 0 Reruns			Show all details / Hide all details
Result 	Test	Duration	Links
Passed	tests/step-defs/test_ct01_valid_login.py::test_ct01_valid_login	00:00:20	
Passed	tests/step-defs/test_ct02_labels_create_account.py::test_ct02_labels_create_account	00:00:34	
Passed	tests/step-defs/test_ct03_create_account.py::test_ct03_create_account	00:00:50	
Passed	tests/step-defs/test_ct04_labels_create_opportunity.py::test_ct04_labels_create_opportunity	00:00:34	
Passed	tests/step-defs/test_ct05_create_opportunity.py::test_ct05_create_opportunity	00:00:49	
Passed	tests/step-defs/test_ct06_labels_create_lead.py::test_ct06_labels_create_lead	00:00:41	
Passed	tests/step-defs/test_ct07_create_lead.py::test_ct07_create_lead	00:01:03	
Passed	tests/step-defs/test_ct08_edit_account.py::test_ct08_edit_account	00:00:56	
Passed	tests/step-defs/test_ct09_edit_opportunity.py::test_ct09_edit_account	00:00:56	
Passed	tests/step-defs/test_ct10_edit_lead.py::test_ct10_edit_lead	00:01:06	

RESULTADOS

Os resultados obtidos evidenciam os benefícios da automação de testes em termos de eficiência de tempo.



Tempo total no teste automatizado: 7:49
Tempo total no teste manual: 41:10



CONCLUSÕES

Resumo, ferramentas e técnicas utilizadas

- Automação de 10 casos de teste funcionais no CRM Salesforce.
- Fluxo completo de usuário em três objetos distintos;
- Execução no navegador Google Chrome com geração de relatório HTML detalhado;
- Aqua IDE, Python, Selenium, pytest e pytest-bdd;
- Adoção do POM para organização e reutilização de código.

Benefícios da automação

- Viabilidade na criação e manutenção de testes escalonáveis;
- Testes automatizados aproximadamente 5 vezes mais rápidos que os manuais;
- Redução de custos e reuso dos testes desenvolvidos.

CONCLUSÕES

Contribuições do trabalho

- Design estruturado: Reutilização de funções e separação entre dados e funcionalidades;
- Ambiente otimizado: Simplificação na escrita e execução de testes;
- Prova de conceito: Ampliação de referências técnicas sobre automação no Salesforce.

Trabalhos Futuros

- Integração dos testes automatizados em pipelines CI/CD.
- Desenvolvimento de testes para versões mobile e tablet do sistema.

BIBLIOGRAFIA PRINCIPAL

BERNARDO, P. C.; KON, F. A Importância dos Testes Automatizados. [s.l.]: Engenharia de Software Magazine, 2008. Disponível em: <<https://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf>>. Acesso em: 03 abr. 2024.

CARVALHO, L.. Automação de Testes Web com Selenium Webdriver e Python. [s.l.]: Udemy, mar. 2023. Disponível em: <<https://www.udemy.com/course/automacao-de-testes-selenium-webdriver-python>>. Acesso em: 04 maio 2024. Getting started. Disponível em: <https://www.selenium.dev/documentation/webdriver/getting_started/>. Acesso em: 20 out. 2024.

MYERS, G. J.; BADGETT, T.; SANDLER, C. The Art of Software Testing. 3. ed. Hoboken, New Jersey: John Wiley & Sons, Inc, 2012. E-book. Disponível em: <<https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf>>. Acesso em: 23 mar. 2024.

OKKEN, B. Python Testing with Pytest. Simple, Rapid, Effective, and Scalable. 1. ed. Raleigh: The Pragmatic Programmers, LLC, 2017. E-book. Disponível em: <<http://www.3testing.com/doc/23.pdf>>. Acesso em: 03 abr. 2024.

SAINI, Manish. Page Object Model and Page Factory in Selenium Python. Disponível em: <<https://www.browserstack.com/guide/page-object-model-in-selenium-python#toc0>>. Acesso em: 01 abr. 2024.

SOMMERVILLE, I. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: <<https://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf>>. Acesso em: 03 abr. 2024.

FIM