Documentação do Projeto de Testes Automatizados

Desenvolvido por Nathalia Koetz e Guilherme Acosta

Este documento apresenta o projeto de testes automatizados desenvolvido pela nossa dupla como parte da atividade avaliativa 2. Nosso objetivo principal foi criar uma aplicação web simples e, em seguida, desenvolver testes unitários e testes funcionais (de sistema) para garantir sua correta operação e robustez.

---

1. Objetivo do Projeto

O objetivo central foi aplicar os conceitos de testes automatizados, utilizando uma **linguagem de programação (Python)**, para validar uma aplicação web. Especificamente, focamos em dois níveis de teste:

- **Testes Unitários:** Para verificar componentes isolados da lógica de negócio da aplicação.
- **Testes de Sistema (Funcionais):** Para simular a interação de um usuário real com a interface da aplicação no navegador.

---

2. Aplicação Desenvolvida: Gerenciador de Tarefas Simples

Para a prática, optamos por desenvolver um **Gerenciador de Tarefas simples**. Essa escolha foi estratégica por ser uma aplicação que nos permitiu criar facilmente cenários para ambos os tipos de testes.

Funcionalidades Implementadas:

- **Adicionar Tarefa:** O usuário pode inserir um texto e adicionar uma nova tarefa à lista.
- **Marcar/Desmarcar Tarefa:** É possível alternar o status de uma tarefa entre "concluída" e "não concluída". Visualmente, tarefas concluídas são riscadas.
- **Excluir Tarefa:** O usuário pode remover tarefas da lista.

Tecnologias Utilizadas:

- **Frontend:** HTML e CSS básico para a interface de usuário.
- **Backend: Flask (Python)**, um microframework web leve e flexível.
- **Dados:** As tarefas são armazenadas em uma lista simples em memória (para a finalidade do projeto).

---

3. Desenvolvimento dos Testes Automatizados

3.1. Testes Unitários

Ferramenta: Utilizamos o módulo `unittest`, nativo do Python, por sua simplicidade e eficácia para testes de unidade.

Cenários de Teste (Divisão da Dupla):

- **[Guilherme Acosta]: Validação de Entrada de Tarefas**
  - Testamos a função `validate_task` que verifica se o texto da tarefa é válido antes de ser adicionado.
  - **Casos de Teste:** Validamos entradas vazias, entradas com apenas espaços, entradas que excedem o limite de caracteres e entradas válidas. Isso garante que apenas dados consistentes sejam processados pela aplicação.
- **[Nathalia Koetz]: Manipulação de Estado da Tarefa (Simulação)**
  - Testamos a lógica de marcação de tarefas como concluídas ou desmarcadas, simulando a alteração de status em uma lista de tarefas.
  - **Casos de Teste:** Verificamos se uma tarefa existente pode ser marcada ou desmarcada corretamente, e se tentar manipular uma tarefa inexistente não causa erros nem altera outras tarefas.

3.2. Testes de Sistema (Funcionais)

Ferramenta: Utilizamos **Selenium WebDriver** (com Python) em conjunto com o `unittest`, que é a ferramenta padrão da indústria para automação de testes de interface de usuário (UI)

Cenários de Teste (Divisão da Dupla):

- **[Nathalia Koetz]: Adicionar e Verificar Tarefa**
  - **Caso de Teste:** Automatizamos o processo de digitar um texto no campo de entrada, clicar no botão "Adicionar" e, em seguida, verificamos se a tarefa aparece corretamente na lista de tarefas exibida na página. Isso valida o fluxo de adição e a renderização da UI.
- **[Guilherme Acosta]: Marcar e Desmarcar Tarefa**
  - **Caso de Teste:** O teste adiciona uma tarefa, depois simula o clique no botão para marcá-la como concluída, verificando se o estilo visual (texto riscado, por exemplo) é aplicado. Em seguida, desmarca a tarefa e verifica se o estilo é removido. Isso valida a interação do usuário com o status da tarefa.
- **(Cenário Extra): Excluir Tarefa**
  - Demonstramos também a capacidade de excluir uma tarefa, automatizando o clique no botão "Excluir" e confirmando que a tarefa desaparece da lista.

---

4. Estrutura do Projeto e Execução

Organizamos o código em uma estrutura clara, separando a aplicação dos testes em pastas dedicadas (`app/` e `tests/`).

Para executar o projeto, siga os passos abaixo (também detalhados no `README.md`):

1. **Instalar Dependências:** Instalar `Flask` e `selenium` via `pip install -r requirements.txt`.
2. **Configurar WebDriver:** Baixar o `chromedriver` (ou driver do navegador escolhido) compatível com seu navegador e colocá-lo na raiz do projeto.
3. **Iniciar a Aplicação Flask:** Rodar `python app/app.py` em um terminal.
4. **Executar Testes Unitários:** Em outro terminal, ir para `tests/unit/` e rodar `python test_logica_negocio.py`.
5. **Executar Testes de Sistema:** Com a aplicação Flask rodando, em um terminal, ir para `tests/system/` e rodar `python test_fluxo_sistema.py`.

---

5. Conclusão

Este projeto nos permitiu vivenciar o ciclo completo de desenvolvimento de uma pequena aplicação com a aplicação de testes automatizados em diferentes níveis. Entendemos a importância de ter testes unitários para a validação da lógica de negócio e testes de sistema para garantir a experiência do usuário, resultando em um software mais confiável e de fácil manutenção.