Documento Técnico – Decisões Arquiteturais sobre Cucumber e Page Object Model

1. Objetivo

Este documento justifica, de forma técnica, as decisões adotadas na arquitetura de testes do projeto MOUTSTI_TESTE no que diz respeito à **não utilização das abordagens Cucumber (BDD)** e **Page Object Model (POM)**. A análise considera fatores como performance, manutenibilidade, clareza e alinhamento com as capacidades nativas do Cypress.

---

2. Cucumber (BDD) – Por que não utilizamos

2.1 Visão Geral

Cucumber é uma ferramenta de Behavior-Driven Development (BDD) que utiliza a linguagem Gherkin para descrever cenários de teste em uma estrutura de passos: `Given`, `When`, `Then`. Embora amplamente adotado em projetos com foco colaborativo, essa abordagem não é recomendada para projetos predominantemente técnicos e de automação como o presente.

2.2 Motivos Técnicos para não utilizar Cucumber

2.2.1 Adição de Camadas Desnecessárias

O Cucumber introduz camadas adicionais entre a descrição do teste e a implementação real, resultando em manutenção mais complexa. Os arquivos `.feature` precisam de correspondência exata com os arquivos de step definitions, aumentando o risco de inconsistência.

2.2.2 Custo de Performance

A presença de um parser adicional (Gherkin) afeta o tempo de carregamento e execução. Em testes contínuos e pipelines automatizadas, essa latência acumulada pode se tornar significativa.

2.2.3 Baixo Valor para Testes Técnicos

Quando os testes são elaborados por engenheiros de qualidade ou desenvolvedores, a linguagem natural não oferece valor adicional. Ao contrário, oculta detalhes técnicos importantes, dificultando o troubleshooting.

2.2.4 Debugging Prejudicado

Os erros apontados pelo Cucumber ocorrem nos steps genéricos, sem indicar claramente o ponto da falha no código base. Isso dificulta a depuração e reduz a eficiência da resolução de problemas.

2.3 Conclusão sobre Cucumber

O uso do Cypress, que já possui uma DSL rica e expressiva (`cy.get()`, `cy.request()`, `cy.intercept()`, etc.), torna o Cucumber redundante e prejudicial em termos de desempenho e clareza.

---

3. Page Object Model (POM) – Por que não utilizamos

3.1 Visão Geral

POM é um padrão de design amplamente usado para encapsular a lógica de páginas da aplicação em classes reutilizáveis. Embora útil em contextos específicos, sua aplicação no Cypress requer análise crítica.

3.2 Motivos Técnicos para não utilizar o POM tradicional

3.2.1 Introdução de Acoplamento Desnecessário

Com o tempo, a manutenção de objetos de página torna-se cara. Mudanças simples no front-end exigem alterações em múltiplos arquivos, aumentando o esforço de manutenção e o risco de regressões.

3.2.2 Incompatibilidade com o Paradigma do Cypress

Cypress adota uma abordagem assíncrona e encadeada (chainable commands). Forçar o uso de objetos orientados a classe conflita com esse modelo e pode levar a práticas antinaturais ou gambiarras.

3.2.3 Impacto Direto na Performance

A abstração excessiva introduz delays, indireções e testes menos legíveis. Cada função encapsulada pode conter chamadas internas que dificultam a visualização do fluxo real.

3.2.4 Testes Frágeis e de Difícil Rastreabilidade

Com o tempo, grandes classes de page objects acumulam responsabilidades, tornando o código pouco coeso e dificultando a identificação de falhas.

3.3 Alternativa Adotada: Component Object Model (COM)

Optamos por adotar o **Component Object Model**, onde funções utilitárias são agrupadas por **comportamento ou componente reutilizável** (e não por tela), promovendo:

- Modularidade leve
- Reutilização real e controlada
- Compatibilidade com o estilo declarativo do Cypress

---

4. Benefícios das Abordagens Adotadas

Critério	Abordagem Adotada	Resultado
Performance	Cypress nativo sem BDD/POM	Alta
Curva de aprendizado	Direta e sem sobrecarga	Baixa
Clareza dos testes	Alta legibilidade	Alta
Facilidade de debugging	Cypress Runner + `cy.api()`	Alta
Escalabilidade	Modular por contexto (CTAF)	Alta

---

5. Considerações Finais

As decisões de não adotar Cucumber e Page Object Model foram tomadas com base em critérios objetivos e mensuráveis. A estrutura atual privilegia clareza, velocidade, rastreabilidade e facilidade de manutenção. Esse direcionamento torna a framework mais eficiente, resiliente e preparada para crescer de forma sustentável.