Análise de Smells

1.0 - Uso de “if” e “for” dentro de testes:

```
// O teste tem um loop e um if, tornando-o complexo e menos claro.
for (const user of todosOsUsuarios) {
  const resultado = userService.deactivateUser(user.id);
  if (!user.isAdmin) {
    // Este expect só roda para o usuário comum.
    expect(resultado).toBe(true);
    const usuarioAtualizado = userService.getUserById(user.id);
    expect(usuarioAtualizado.status).toBe('inativo');
  } else {
    // E este só roda para o admin.
    expect(resultado).toBe(false);
  }
}
```

Por que é um mau cheiro:

O teste mistura múltiplos cenários (usuário comum e admin) em um único caso de teste. Isso quebra o princípio da atomicidade, dificultando entender o que exatamente falhou quando um dos cenários dá erro.

Risco: Se uma das condições falhar, o outro caso pode mascarar o erro. Dificulta manutenção e leitura, especialmente em refatorações futuras.

2.0 - Testes dependentes de formatação específica:

```
▷ Execute Playwright Test | Add to List
test('deve gerar um relatório de usuários formatado', () => {
  const usuario1 = userService.createUser('Alice', 'alice@email.com', 28);
  userService.createUser('Bob', 'bob@email.com', 32);

  const relatorio = userService.generateUserReport();

  // Se a formatação mudar (ex: adicionar um espaço, mudar a ordem), o teste quebra.
  const linhaEsperada = `ID: ${usuario1.id}, Nome: Alice, Status: ativo\n`;
  expect(relatorio).toContain(linhaEsperada);
  expect(relatorio.startsWith('--- Relatório de Usuários ---')).toBe(true);
});
```

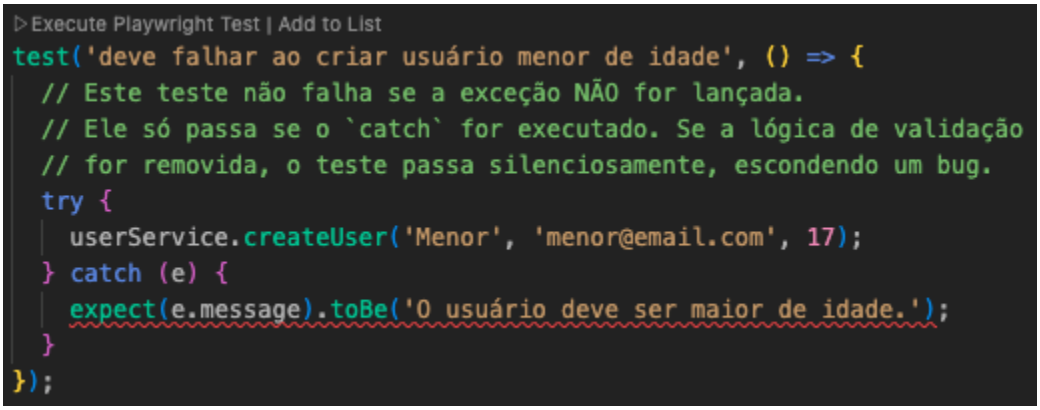
Por que é um mau cheiro:

O teste depende da formatação literal do relatório (espaços, quebras de linha, ordem dos campos).

Pequenas mudanças na string podem quebrar o teste, mesmo que o comportamento continue correto.

Risco: Testes quebram com mudanças cosméticas. Gera falsos negativos, diminuindo a confiança da equipe nos testes automatizados.

3.0 - Teste que passa mesmo quando deveria falhar:



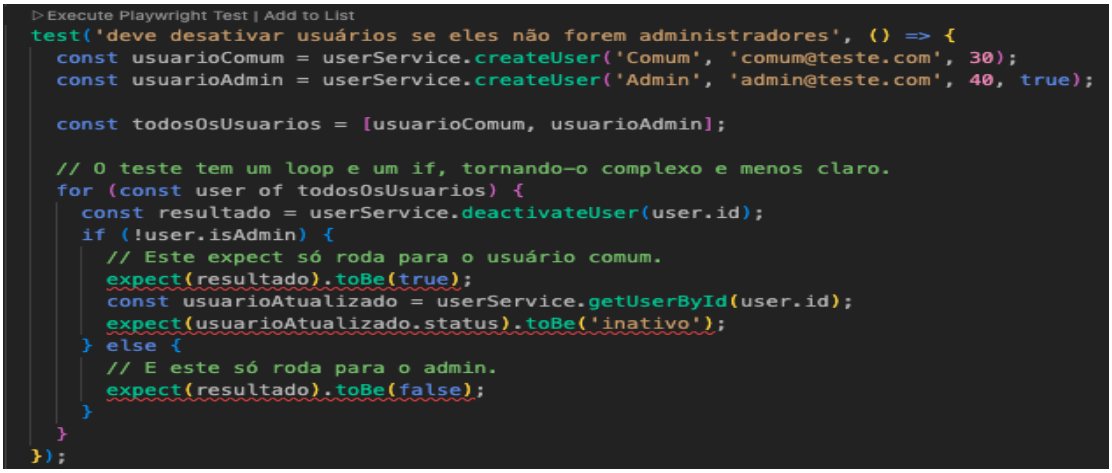
Por que é um mau cheiro:

Se o erro não for lançado, o teste passa sem executar nenhuma asserção portanto, falso positivo.

Risco: A validação pode ser removida do código e o teste continuará passando. Oculta defeitos reais e dá falsa sensação de segurança.

Processo de Refatoração:

Antes:



Depois:

```
▷ Execute Playwright Test | Add to List
test('deve desativar um usuário comum com sucesso', () => {
  // Arrange
  const usuario = userService.createUser('Comum', 'comum@teste.com', 30);

  // Act
  const resultado = userService.deactivateUser(usuario.id);

  // Assert
  expect(resultado).toBe(true);
  const usuarioAtualizado = userService.getUserById(usuario.id);
  expect(usuarioAtualizado.status).toBe('inativo');
});
```

```
▷ Execute Playwright Test | Add to List
test('não deve desativar um usuário administrador', () => {
  // Arrange
  const admin = userService.createUser('Admin', 'admin@teste.com', 40, true);

  // Act
  const resultado = userService.deactivateUser(admin.id);

  // Assert
  expect(resultado).toBe(false);
  const usuarioAtualizado = userService.getUserById(admin.id);
  expect(usuarioAtualizado.status).toBe('ativo');
});
```

Problemas: Loop e if dentro do teste, teste longo e genérico e falta de clareza na intenção.

Soluções Aplicadas: Dividido em dois testes separados, um para cada cenário: usuário comum e admin. Substituído por testes atômicos e descritivos. Cada teste tem nome autoexplicativo e segue o padrão AAA.

Relatório da Ferramenta ESLint:

```
✘ educbank@macbook-pro-vinicius-educbank ~/my-projects/test-smelly  main ± npx eslint .

/Users/educbank/my-projects/test-smelly/test/userService.smelly.test.js
  44:9  error    Avoid calling `expect` conditionally`  jest/no-conditional-expect
  46:9  error    Avoid calling `expect` conditionally`  jest/no-conditional-expect
  49:9  error    Avoid calling `expect` conditionally`  jest/no-conditional-expect
  73:7  error    Avoid calling `expect` conditionally`  jest/no-conditional-expect
  77:3  warning  Tests should not be skipped            jest/no-disabled-tests
  77:3  warning  Test has no assertions                 jest/expect-expect

✖ 6 problems (4 errors, 2 warnings)
```

Comentário sobre a ferramenta:

O ESLint, com o plugin eslint-plugin-jest, detectou automaticamente: Uso de estruturas proibidas (if, for) dentro de testes, uso inadequado de try/catch em asserts, sugestão de uso do matcher Jest toThrow(). Essa análise estática automatizou a detecção de padrões ruins sem necessidade de execução dos testes, facilitando a manutenção preventiva da base de código.

Conclusão:

A refatoração mostrou como testes limpos tornam o código mais confiável, legível e sustentável.

Aplicando boas práticas (AAA, atomicidade, ausência de lógica condicional e asserts precisos): Facilita a compreensão do que o teste realmente válido, evita falsos positivos/negativos, aumentando a confiança na suíte, permite refatorações seguras no código de produção. Além disso, o uso de ferramentas de análise estática como o ESLint garante detecção precoce de más práticas, funcionando como uma camada de defesa contínua da qualidade.