**Sistema de coleta de dados de Estações Meteorológicas**

Logotipo

O conteúdo gerado por IA pode estar incorreto.

**2025.**

# 📘 Testes de Integração - Equipe SYNC

## **Guia Prático para Equipes de Desenvolvimento**

## **Introdução**

### **O que são Testes de Integração** Os testes de integração verificam se diferentes módulos ou serviços do seu aplicativo funcionam bem juntos. Diferentemente dos testes unitários que isolam componentes individuais, os testes de integração avaliam as interações entre componentes integrados e identificam problemas que só aparecem quando esses componentes trabalham em conjunto. Em nosso contexto, os testes de integração validam: - Comunicação entre diferentes camadas da aplicação - Interações com bancos de dados e serviços externos (como MongoDB) - Fluxos completos de dados através de múltiplos componentes - Comportamento correto das interfaces entre módulos

### **Benefícios para a Equipe** A abordagem de testes de integração descrita neste documento traz diversos benefícios: Colaboração aprimorada: PO, desenvolvedores e Analista de testes trabalham juntos desde o início Documentação viva: Cenários em Gherkin servem como documentação executável e atualizada Confiança nas entregas: Maior segurança para realizar mudanças e evoluções no código Previsibilidade: Redução de surpresas de última hora que poderiam comprometer a entrega

## **Fluxo de Trabalho**

### **Visão Geral do Processo** O processo de testes de integração está totalmente integrado ao ciclo de desenvolvimento ágil:

┌───────────────────────────────┐  
│ Planejamento Inicial │  
│ • PO define tarefas com flag │  
│ • Marcadas no Jira como │  
│ "Requer Teste de Integração" │  
└──────────────┬────────────────┘  
 │  
 ▼  
┌───────────────────────────────┐  
│ Escrita dos Casos Gherkin │  
│ • Analista de Testes escreve │  
│ os cenários no Dia 1 │  
│ • Baseados nos critérios PO │  
└──────────────┬────────────────┘  
 │  
 ▼  
┌───────────────────────────────┐  
│ Desenvolvimento + Testes │  
│ • Dev implementa a feature │  
│ • Escreve testes com base │  
│ nos cenários definidos │  
└──────────────┬────────────────┘  
 │  
 ▼  
┌───────────────────────────────┐  
│ Revisão e Validação de PR │  
│ • QA revisa se os testes │  
│ cobrem todos os Gherkins │  
│ • Aprova ou solicita ajustes │  
└───────────────────────────────┘

## **Processo Detalhado**

### **Planejamento Inicial** Quando: Planejamento da sprint Participantes: Product Owner (PO) Objetivo: Identificar quais tarefas da sprint necessitam de testes de integração e priorizar aquelas com maior impacto técnico e funcional e marcar como flag no jira.

### **Escrita dos casos** Responsável: Analista de Testes Formato: Linguagem Gherkin seguindo o padrão BDD (Behavior-Driven Development) Conteúdo: Cenários que cobrem tanto fluxos positivos quanto negativos e de bordas Exemplo:

Feature: Processamento de medidas do MongoDB  
 Como um sistema de monitoramento  
 Eu quero processar dados recebidos do MongoDB  
 Para que as medições sejam registradas corretamente nas estações  
  
 Scenario: Recebimento de dados no Mongo  
 Given que a estação e os parâmetros estão cadastrados  
 When o sistema receber os dados do Mongo  
 Then o sistema deve conseguir cadastrar todas as medidas nas estações  
  
 Scenario: Estação ou parâmetros não cadastrados  
 Given que a estação ou os parâmetros não estejam cadastrados  
 When o sistema receber os dados do Mongo  
 Then o sistema deve ignorar as medidas recebidas

Armazenamento: Arquivos .feature no diretório correspondente ao caso de uso

### **Desenvolvimento dos Testes** Responsável: Desenvolvedor da tarefas Momento: Em paralelo com o desenvolvimento da funcionalidade Abordagem: Automação baseada nos cenários Gherkin definidos previamente Critério de Conclusão: A funcionalidade só é considerada pronta quando os testes de integração estão implementados e passando

### **Revisão e Aprovação** Gatilho: Pull Request aberto pelo desenvolvedor Revisor: Analista de Testes Critérios de Verificação: - Todos os cenários definidos foram implementados - Os testes cobrem fluxos esperados e casos de exceção - A estrutura e qualidade dos testes seguem os padrões da equipe Resultado: Aprovação ou solicitação de ajustes

## **Integração com CI** Execução automatizada: Os testes de integração automatizados são executados apenas nas branches de pré-produção e produção, garantindo validação completa antes da implantação.

## **Estrutura e Organização**

## **Back-end**

tests/  
 └── integration/  
 ├── config/  
 │ ├── seeds/ # Funções para gerar dados de teste  
 │ └── setup/ # Configuração do ambiente de teste  
 └── application/<nome-da-entidade>/ # Pasta para cada entidade  
 ├── <nome-do-usecase>.test.ts # Implementação do teste  
 └── <nome-do-usecase>.feature # Especificação em Gherkin

#### Detalhamento dos Diretórios: config/seeds/: Contém funções que preparam os dados necessários para os testes. Cada seed deve utilizar os use cases reais do sistema para criar os dados. config/setup/: Possui arquivos para inicialização do banco de dados de teste, limpeza entre os testes e configuração do ambiente. application/&lt;nome-da-entidade&gt;/: Diretórios organizados por domínio (ex: measures, station, parameters). \*.test.ts: Arquivos com os testes de integração, nomeados conforme o use case testado. \*.feature: Arquivo contendo os casos de testes escritos em linguagem Gherkin.

### **Front-end**

src/  
 └── pages/  
 └── NomeDaFuncionalidade/  
 └── NomeDoComponente/  
 └── NomeDoComponente.int.test.tsx

#### Convenções de Nomenclatura: Para facilitar a identificação dos diferentes tipos de teste, utilizamos o sufixo .int.test.tsx para testes de integração, distinguindo-os dos testes unitários (.unit.test.tsx).

## **Configuração do Ambiente**

## **Arquivos de Configuração:** SetupIntegration.ts: Inicializa o TypeORM com o banco de dados de teste. DatabaseCleaner.ts: Limpa todas as tabelas do banco entre os testes. TeardownIntegration.ts: Encerra as conexões após os testes.

## **Execução dos Testes**

## # Executa todos os testes de integração npm run test:integration

## **Dependências e Versões** Para garantir a compatibilidade e funcionamento correto dos testes, utilize as seguintes versões das bibliotecas principais:

### **Back-end**

{  
 "dependencies": {  
 "jest-cucumber": "^3.0.1",  
 "cucumber": "^6.0.7",  
 "typeorm": "^0.3.20",  
 "pg": "^8.11.3",  
 "testcontainers": "^10.0.0",  
 "jest": "^29.7.0",  
 "@types/jest": "^29.5.12",  
 "ts-jest": "^29.1.2"  
 }  
}

### **Front-end**

### { "devDependencies": { "jest-cucumber": "^3.0.1", "@testing-library/jest-dom": "^5.17.0", "@testing-library/react": "^13.4.0", "@testing-library/user-event": "^14.4.3", "jest": "^29.7.0", "jest-environment-jsdom": "^29.7.0", "ts-jest": "^29.3.2", "@types/jest": "^29.5.14" } }

## **Anexos e Exemplos**

### **Exemplo de Teste de Integração Back-end**

# measures.feature  
Feature: Processamento de medidas do MongoDB  
 Como um sistema de monitoramento  
 Eu quero processar dados recebidos do MongoDB  
 Para que as medições sejam registradas corretamente nas estações  
  
 Scenario: Recebimento de dados no Mongo  
 Given que a estação e os parâmetros estão cadastrados  
 When o sistema receber os dados do Mongo  
 Then o sistema deve conseguir cadastrar todas as medidas nas estações  
  
 Scenario: Estação ou parâmetros não cadastrados  
 Given que a estação ou os parâmetros não estejam cadastrados  
 When o sistema receber os dados do Mongo  
 Then o sistema deve ignorar as medidas recebidas

// measures.test.ts  
import { defineFeature, loadFeature } from 'jest-cucumber';  
import { setupDatabase } from '../config/setup/database';  
import { seedStation } from '../config/seeds/station';  
import { MeasuresRepository } from '../../src/repositories/MeasuresRepository';  
  
// Carrega o arquivo .feature correspondente  
const feature = loadFeature('./tests/integration/application/measures/measures.feature');  
  
defineFeature(feature, (test) => {  
 beforeAll(async () => {  
 await setupDatabase();  
 await seedStation();  
 });  
  
 test('Recebimento de dados no Mongo', ({ given, when, then }) => {  
 let estacao, parametros, resultado;  
   
 given('que a estação e os parâmetros estão cadastrados', async () => {  
 // Configuração do cenário  
 estacao = await seedStation({ nome: 'Estação Teste' });  
 parametros = await seedParametros({ estacaoId: estacao.id });  
 });  
  
 when('o sistema receber os dados do Mongo', async () => {  
 // Ação principal do teste  
 const measuresService = new MeasuresService(new MeasuresRepository());  
 resultado = await measuresService.processarDadosDoMongo({  
 estacaoId: estacao.id,  
 parametrosId: parametros.id,  
 valor: 25.5  
 });  
 });  
  
 then('o sistema deve conseguir cadastrar todas as medidas nas estações', () => {  
 // Verificação do resultado  
 expect(resultado.success).toBeTruthy();  
 expect(resultado.medidasCadastradas).toBeGreaterThan(0);  
 });  
 });  
  
 test('Estação ou parâmetros não cadastrados', ({ given, when, then }) => {  
 let dadosRecebidos, resultado;  
   
 given('que a estação ou os parâmetros não estejam cadastrados', () => {  
 // Configuração do cenário com estação inexistente  
 dadosRecebidos = {  
 estacaoId: 'estacao-inexistente',  
 parametrosId: 'parametros-validos',  
 valor: 30.2  
 };  
 });  
  
 when('o sistema receber os dados do Mongo', async () => {  
 // Ação principal do teste  
 const measuresService = new MeasuresService(new MeasuresRepository());  
 resultado = await measuresService.processarDadosDoMongo(dadosRecebidos);  
 });  
  
 then('o sistema deve ignorar as medidas recebidas', () => {  
 // Verificação do resultado  
 expect(resultado.ignored).toBeTruthy();  
 expect(resultado.medidasCadastradas).toBe(0);  
 });  
 });  
});

### **Exemplo de Teste de Integração Front-end**

# Dashboard.feature  
Feature: Visualização de medições na dashboard  
 Como um usuário do sistema  
 Eu quero visualizar as medições mais recentes na dashboard  
 Para que eu possa monitorar os dados em tempo real  
  
 Scenario: Exibição de medições na dashboard  
 Given que o usuário está autenticado no sistema  
 And que existem medições disponíveis para exibição  
 When o usuário acessa a página de dashboard  
 Then o sistema deve exibir as medições mais recentes  
  
 Scenario: Filtragem de medições por estação  
 Given que o usuário está na página de dashboard  
 And que existem medições de múltiplas estações  
 When o usuário seleciona uma estação específica no filtro  
 Then o sistema deve exibir apenas as medições da estação selecionada

// Dashboard.int.test.tsx  
import { defineFeature, loadFeature } from 'jest-cucumber';  
import { render, screen, waitFor, within } from '@testing-library/react';  
import userEvent from '@testing-library/user-event';  
import { Dashboard } from './Dashboard';  
import { mockApiResponse } from '../../test/mocks/api';  
  
// Carrega o arquivo .feature correspondente  
const feature = loadFeature('./src/pages/Dashboard/Dashboard.feature');  
  
defineFeature(feature, (test) => {  
 test('Exibição de medições na dashboard', ({ given, when, then }) => {  
 given('que o usuário está autenticado no sistema', () => {  
 // Configuração do cenário com mock de autenticação  
 mockApiResponse('auth', { authenticated: true, user: { name: 'Teste' } });  
 });  
  
 given('que existem medições disponíveis para exibição', () => {  
 // Mock de dados de medições  
 mockApiResponse('measures', {  
 data: [  
 { id: 1, estacao: 'Estação A', parametro: 'Temperatura', valor: 25.5 },  
 { id: 2, estacao: 'Estação B', parametro: 'Umidade', valor: 80 }  
 ]  
 });  
 });  
  
 when('o usuário acessa a página de dashboard', async () => {  
 // Renderiza o componente  
 render(<Dashboard />);  
 // Aguarda carregamento  
 await waitFor(() => screen.getByTestId('dashboard-loaded'));  
 });  
  
 then('o sistema deve exibir as medições mais recentes', async () => {  
 // Verifica se os dados foram exibidos corretamente  
 const medicoesContainer = screen.getByTestId('medicoes-container');  
 expect(within(medicoesContainer).getByText('Estação A')).toBeInTheDocument();  
 expect(within(medicoesContainer).getByText('25.5')).toBeInTheDocument();  
 expect(within(medicoesContainer).getByText('Estação B')).toBeInTheDocument();  
 expect(within(medicoesContainer).getByText('80')).toBeInTheDocument();  
 });  
 });  
});