📄 PLANO DE TESTES – Projeto SaudeMais

Versão: 1.0
Data: *27/11/2025*
Responsável: Thiago Marinho

1. Introdução

Este relatório apresenta o plano de testes unitários desenvolvidos para o sistema **SaudeMais**, utilizando **JUnit 5** e **Mockito** para validar regras de negócio e garantir confiabilidade nas funcionalidades internas do sistema.

Como o projeto contém interface gráfica Swing e não possui banco de dados real, os testes se concentram exclusivamente nas **regras internas**, isolando dependências por meio de mocks.

2. Objetivos dos Testes

- Garantir que regras internas funcionem corretamente.
- Validar comportamentos de métodos sem depender da interface gráfica.
- Simular dependências usando Mockito para permitir testes mais rápidos e controlados.
- Verificar interações entre classes do domínio e serviços.
- Assegurar que regressões sejam detectadas em futuras alterações do sistema.

3. Escopo dos Testes

Incluído

- Testes unitários de regras de negócio.
- Testes de lógica presente em classes auxiliares extraídas do código principal.
- Testes que simulam dependências via Mockito.
- Teste REAL baseado no código existente (`ConsultaLookupService`).

Excluído

- Interface gráfica Swing (JFrame, JTable, JButton).
- Diálogos Swing (`JOptionPane`).
- Persistência real em banco (não existe no projeto).
- Testes de integração.

4. Ferramentas Utilizadas

- **Java 17**
- **JUnit 5 (Jupiter)**
- **Mockito 5**
- **Maven**
- **NetBeans / IntelliJ / Eclipse** (qualquer suporte a Maven)

5. Arquitetura de Testes

A estrutura utilizada segue o padrão Maven:

src/main/java/... → código de produção

src/test/java/... → **código de testes**

Cada teste foi criado isolando o comportamento da classe alvo.

6. Casos de Teste (Plano Completo)

Abaixo estão listados os testes realizados.

◇ 6.1 – Testes Inventados (didáticos) – DoseCalculator

6.1.1 Objetivo

Validar uma classe utilitária responsável por cálculos simples de doses de medicação.

6.1.2 Classe Testada

`DoseCalculator`

6.1.3 Casos de Teste

ID	Caso de Teste	Dado de Entrada	Resultado Esperado
TC-DC-01	Cálculo normal	3 vezes/dia × 4 dias	12
TC-DC-02	Zerando valores	0 × 10	0
TC-DC-03	Strings vazias	"" e "5"	0
TC-DC-04	Valor inválido	"abc"	NumberFormatException
TC-DC-05	Valor negativo	-1	IllegalArgumentException

◇ 6.2 – Testes Inventados com Mockito – DoseService + DoseRepository

6.2.1 Objetivo

Validar o comportamento de um serviço que depende de um repositório externo, simulando esse repositório com Mockito.

6.2.2 Classe Testada

`DoseService`

6.2.3 Dependência Mockada

`DoseRepository` (Mockito)

6.2.4 Casos de Teste

ID	Caso de Teste	Entrada	Comportamento Simulado	Resultado Esperado
TC-DS-01	Cálculo correto	4 dias, “Dipirona”	getVezesAoDia → 3	12
TC-DS-02	Erro de dias negativos	-5	—	IllegalArgumentException
TC-DS-03	Verificar chamada do Repository	qualquer valor	—	verify(repo).getVezesAoDia(..)

◇ 6.3 – TESTE REAL (extraído do projeto)

✓ Baseado diretamente no código do projeto

✓ Utiliza Mockito

✓ Não depende da GUI

6.3 – Teste REAL – ConsultaLookupService

6.3.1 Origem da Regra

A regra existe dentro de `ButtonEditor`, linha:

service.listar().stream()

.filter(c -> c.getId() == idSelecionado)

.findFirst().orElse(null);

Extraímos essa regra para a classe:

`ConsultaLookupService`

6.3.2 Objetivo

Garantir que o sistema seja capaz de localizar uma consulta pelo ID usando `IConsultaService`.

6.3.3 Casos de Teste

ID	Caso de teste	Entrada	Simulação	Resultado Esperado
TC-CL-01	Consulta existente	id = existente	listar() retorna lista com itens	Consulta encontrada

TC-CL-02	Consulta inexistente	id = 999	listar() retorna lista vazia	Retornar null
TC-CL-03	Verificação de Uso da Dependência	qualquer valor	—	verify(mockService).listar()

7. Evidências Esperadas

Para casos de sucesso:

- Asserções positivas (`assertEquals`, `assertNotNull`).
- Chamada correta do mock (`verify()`).
- Execução sem exceções inesperadas.

Para casos de falha planejada:

- Lançamento de `IllegalArgumentException`.
- Lançamento de `NumberFormatException`.
- Retorno `null` quando apropriado.

8. Conclusão

Este plano de testes demonstra:

- Cobertura de regras de negócio reais do SaudeMais.
- Simulação de dependências via Mockito.
- Isolamento completo de interface gráfica e armazenamento.
- Uso correto das melhores práticas de testes unitários (JUnit 5 + Mockito).
- Testes didáticos e testes reais coexistindo no mesmo projeto.

O resultado é uma base sólida para evoluir o sistema com segurança, minimizando regressões e incentivando boas práticas de arquitetura.