PLANO DE TESTE - SISTEMA SIMPLE HEALTH

1. Introdução

1.1. Descrição do Sistema

O Sistema Simple Health é uma solução de gestão de saúde projetada com uma arquitetura de microserviços no back-end e uma aplicação cliente desktop (Front-end) que consome essas APIs. O objetivo é gerenciar recursos de saúde (medicamentos, insumos, estoque) e processos de atendimento ao paciente (cadastro, agendamento).

Componentes Principais:

- **Back-end (APIs):** Composto por microsserviços modulares (Cadastro, Estoque, Agendamento) que realizam operações de CRUD e validam a lógica de negócio.
- **Front-end (Desktop):** Aplicação de interface gráfica JavaFX/Desktop que interage com o usuário final e se comunica com o Back-end via HTTP/APIs.

1.2. Objetivos do Teste

O objetivo central deste plano é garantir a **qualidade, a integridade de dados e a experiência do usuário (UX)** de ponta a ponta em todo o Sistema Simple Health.

Objetivos Específicos (Mantidos e Expandidos):

1. **Funcionalidade da API:** Verificar se todos os endpoints das APIs respondem com os códigos de status HTTP corretos para solicitações válidas e inválidas, e se as respostas seguem os schemas definidos.
2. **Lógica de Negócio:** Validar a lógica de negócio para todas as operações, incluindo a criação, atualização e exclusão de recursos em todos os módulos, garantindo a consistência entre eles.
3. **Integridade de Dados:** Assegurar a integridade e a consistência dos dados no banco de dados após cada operação em todos os microserviços.
4. **Experiência do Usuário (E2E):** Garantir que a aplicação Front-end está livre de defeitos de UI, é usável e realiza a comunicação correta com os serviços de Back-end (fluxo End-to-End).

2. Estratégia de Teste

2.1. Abordagem de Teste (Metodologia)

A metodologia utilizada será baseada em uma Abordagem Ágil/Iterativa, com foco em testes de API e integração desde o início, complementados por testes de Sistema (End-to-End) na fase final do projeto.

Fases do Processo de Teste:

1. **Planejamento e Design:** Revisão da documentação, elaboração da matriz de rastreabilidade (requisitos x casos de teste) e criação dos casos de teste.
2. **Configuração:** Preparação dos ambientes e ferramentas (Postman, Banco de Dados, Aplicação Front-end, Swagger Backend).
3. **Execução (API):** Execução automatizada e manual dos testes de API (Funcionais, Contrato/Schema, Negativos).
4. **Execução (E2E/UI):** Execução de testes no Front-end, validando a interface, a usabilidade e a comunicação com as APIs.
5. **Registro e Rethinking:** Documentação de bugs no Jira e elaboração de relatórios parciais.

2.2. Níveis de Teste

O plano cobrirá os seguintes níveis de teste para garantir a qualidade em todas as camadas da aplicação:

Nível de Teste	Foco Principal	Ferramentas
Testes Unitários	Componentes internos de cada microserviço (classes, métodos).	JUnit, Mockito
Testes de Integração	Comunicação entre os módulos da API	Postman, Rest Assured.
Testes de Sistema (E2E)	Fluxos completos de negócio que envolvem o Front-end e toda a cadeia de microserviços.	Testes manuais
Testes de alinhamento (documentação)	Validar conformidade do sistema com a documentação fornecida	Documentação fornecida e código fonte da aplicação

3. Escopo Detalhado e Funcionalidades a Serem Testadas

O escopo expandido cobre todos os microserviços e a interface do usuário correspondente.

Módulo	Tipo de Recurso	Funcionalidades Críticas de Teste (CRUD + Regras)
Estoque	Pedidos, Medicamentos, Hospitalares, Fornecedores, Estoques, Alimentos, Itens Genéricos.	Criação de Pedido associado a Fornecedor e Itens existentes; **Atualização correta da quantidade em estoque** após entrada/saída; Impossibilidade de deletar Fornecedor com Pedido associado.
Cadastro	Pacientes, Médicos, Convênios, Usuários.	Criação/Atualização com validação de duplicidade (Ex: CPF/CNPJ); Verificação de Status Ativo/Inativo para Agendamento/Uso do sistema; Validação de campos obrigatórios (Ex: CRM do Médico).
Agendamento	Consultas, Exames, Procedimentos, Bloqueio de Agenda.	Agendamento deve respeitar a disponibilidade do recurso (Médico/Sala); Não permitir agendamento em horário de **Bloqueio de Agenda**; Cancelamento deve liberar o horário; **Encaixe** deve ser validado para garantir que não sobreponha agendamento existente; Consulta ao histórico do paciente.
Front-end	Interfaces de Cadastro, Estoque e Agendamento.	Navegação e Fluxo de Telas; Validações de entrada de dados (formatos, obrigatórios); Exibição correta de dados e tratamento visual de erros (mensagens amigáveis); Teste de Usabilidade.

3.1. Tipos de Teste Não-Funcionais

Além dos testes funcionais (que garantem o que o sistema faz), serão aplicados testes não-funcionais (que garantem como o sistema funciona):

- **Teste de Usabilidade (Front-end):** Focado em user experience (UX), garantindo que a navegação é intuitiva e que a informação é apresentada de forma clara.

Critério: 90% dos usuários de teste devem completar a tarefa crítica (ex: agendar uma consulta) em tempo hábil.

- **Teste de Segurança (API):** Verificação de vulnerabilidades conhecidas, como injeção de SQL (se houver consultas diretas), e garantia de que os mecanismos de autenticação/autorização estão em vigor.
- **Teste de Performance/Carga:** (Se aplicável) Medir o tempo de resposta das APIs críticas (ex: Agendar Consulta, Listar Estoque) sob carga simulada.

4. Ambientes, Ferramentas e Configuração

4.1. Ambiente de Teste

1. **Ambiente de Desenvolvimento/Local:** Usado para testes unitários e testes de API iniciais.
    - URL Base das APIs: http://localhost:8081, http://localhost:8082, http://localhost:8083.
    - Banco de Dados: PostgreSQL, MongoDb e Cassandra rodando via Docker.

4.2. Ferramentas de Teste

Categoria	Ferramenta	Propósito Específico
API	**Postman** (Manual/Automação de Coleções) / **Rest Assured** (Automação Java)	Criação e execução de requisições HTTP, verificação de códigos de status, corpo e cabeçalhos.
Front-end	**TestFX**	Automação de testes de UI e End-to-End, simulação de interações do usuário.
Bancos de Dados	**DBeaver**	Validação da integridade e consistência dos dados após operações (CRUD).
Gestão de Testes	**Jira**	Registro de Defeitos, Test Cases e Rastreabilidade.

4.3. Configuração da Máquina de Teste

A máquina de teste deve ser robusta o suficiente para executar o Front-end e, idealmente, simular o ambiente de desenvolvimento local dos 3 serviços ao mesmo tempo.

- **Hardware:** Mínimo 16GB de RAM, Processador i9 11800h, 100GB de armazenamento livre.

- **Software:** Sistema Operacional Linux, Postman instalado, acesso à rede.

5. Análise de Risco e Plano de Mitigação

A Tabela de Risco será expandida para incluir os novos microserviços e o Front-end, detalhando o impacto e o plano de mitigação. A Severidade é calculada como Impacto * Probabilidade.

Categoria	Risco	Impacto (1-5)	Probabilidade (1-5)	Severidade (Max 25)	Plano de Mitigação
Estoque	Falha na atualização da quantidade de um item no estoque após um pedido.	5	4	20 (Crítica)	Implementar Testes de Integração automatizados para o fluxo de Pedido/Estoque. Revisão de Código 4 Olhos no módulo de cálculo.
Agendamento	Agendamento permite sobreposição de horários (conflito de agenda).	5	3	15 (Alta)	Implementar Testes de Regressão automatizados para a validação de horário e para o use case de **Bloqueio de Agenda**.
Geral/Segurança	API vulnerável a ataques de Injeção de Código/SQL.	5	2	10 (Média)	Utilizar frameworks de ORM que previnam injeção por padrão. Realizar Scan de Segurança estático (SAST).

Geral/Front-end	Falha de carregamento da interface (UI crash) devido a erros na resposta da API.	4	3	12 (Alta)	Testes E2E com foco no tratamento de exceções do Front-end (ex: API retorna 500, a GUI não deve quebrar, mas exibir erro amigável).
Cadastro	Permitir cadastro de Paciente/Médico com CPF/CRM duplicado.	4	3	12 (Alta)	Implementar Testes de Validação de unicidade na camada de serviço (Back-end) e de UI (Front-end).

6. Cronograma de Teste

O cronograma será distribuído em fases, priorizando a estabilidade do Back-end antes do Front-end.

Fase	Duração Estimada	Atividades Chave	Entregáveis
Fase 0: Preparação	1 dia	Planejamento, criação e revisão de casos de teste, configuração de ambiente e ferramentas.	Documento de Casos de Teste (Rascunho), Ambientes de Teste prontos.
Fase 1: Back-end Crítico	2 dias	Execução de testes de API (Funcional e Negativo) nos módulos: Estoque (Pedidos/Atualização), Agendamento (Criação/Bloqueio).	Relatório de Teste de API (Postman/Rest Assured), Bugs registrados no Jira.
Fase 2: Integração & UI	3 dias	Execução de testes de integração (Microserviços, DB), Testes E2E (Front-end ↔ Back-end), Testes de Regressão.	Cobertura de Teste de Integração.

Fase 3: Finalização	1 dia	Testes Exploratórios, Testes de Segurança básicos. Elaboração do Relatório Final.	**Relatório Final de Testes** (Google Docs), Recomendação Final .

7. Comunicação e Relatórios

7.1. Ferramentas de Comunicação

- **Comunicação Rápida:** Whatsapp e Discord serão utilizados para esclarecimento de dúvidas urgentes e comunicação em tempo real com a equipe de desenvolvimento..

7.2. Relatório Final de Testes

Ao final do ciclo de testes, um Relatório Final será gerado em formato de documento (Google Docs), contendo:

- Resumo dos testes executados por módulo (Back-end e Front-end).
- Métricas: Número de casos de teste planejados, executados, aprovados e reprovados.
- Lista de Defeitos encontrados (com suas severidades e status atual).
- Cobertura de Regras de Negócio atingida.
- Recomendação final sobre a qualidade da API e do Sistema (se está pronto para produção ou não).

7.3. Formato de Registro de Defeitos

Todos os bugs serão registrados no **Quadro do Jira** com o seguinte formato obrigatório:

- **ID do Defeito:** Identificador único.
- **Título:** Resumo claro e conciso do problema.
- **Componente Afetado:** Ex: Front-end Agendamento, API Estoque, Integração Cadastro-Agendamento.
- **Endpoint / Tela:** Qual endpoint da API ou tela do Front-end foi afetado.
- **Passos para Reproduzir:** Descrição detalhada de como reproduzir o erro.
- **Resultado Esperado:** O que deveria ter acontecido.
- **Resultado Obtido:** O que de fato aconteceu.
- **Evidências:** Prints da tela, payloads da resposta da API (corpo, status, etc.) ou logs.
- **Severidade:** (Crítica, Alta, Média, Baixa).