PLANO DE TESTE - API SIMPLE HEALTH

1. Introdução

1.1. Descrição do Sistema

A API Simple Health é um sistema de back-end projetado para gerenciar recursos de saúde, como medicamentos, alimentos, itens hospitalares e estoque (previsão de expansão futura). A API permite realizar operações de CRUD (Criar, Ler, Atualizar e Excluir) para gerenciar pedidos, fornecedores e os itens em si, simulando um ambiente real de gestão de suprimentos de saúde.

2. Estratégia de Teste

2.1. Objetivos do Teste

O objetivo principal deste plano de teste é garantir a qualidade e a robustez da API Simple Health. Os objetivos específicos são:

- Verificar se todos os endpoints da API respondem com os códigos de status HTTP corretos para solicitações válidas e inválidas.
- Garantir que as respostas da API sigam os schemas definidos na documentação
- Validar a lógica de negócio para todas as operações, incluindo a criação, atualização e exclusão de recursos.
- Assegurar a integridade e a consistência dos dados no banco de dados após cada operação.
- Identificar e documentar quaisquer falhas, inconsistências ou oportunidades de melhoria na API.

2.2. Escopo do Teste

O escopo deste teste cobre todas as funcionalidades de back-end expostas através da API. As seguintes funcionalidades serão testadas:

- **Gerenciamento de Pedidos:** Criar, listar, buscar por ID, atualizar e deletar pedidos.
- **Gerenciamento de Medicamentos:** Criar, listar, buscar por ID, atualizar e deletar medicamentos.
- **Gerenciamento de Itens Hospitalares:** Criar, listar, buscar por ID, atualizar e deletar itens hospitalares.
- **Gerenciamento de Fornecedores:** Criar, listar, buscar por ID, atualizar e deletar fornecedores.
- **Gerenciamento de Estoques:** Criar, listar, buscar por ID, atualizar e deletar estoques.
- **Gerenciamento de Alimentos:** Criar, listar, buscar por ID, atualizar e deletar alimentos.
- **Gerenciamento de Itens Genéricos:** Listar e buscar por ID itens genéricos.

2.3. Níveis de Teste

- **Teste de Integração:** Validar se os diferentes módulos da API (ex: Pedidos e Itens) se comunicam corretamente. Por exemplo, verificar se ao criar um pedido, os itens associados a ele são corretamente referenciados.
- **Teste de Sistema:** Testar a API como um todo para garantir que ela atende a todos os requisitos funcionais e não funcionais definidos na documentação.
- **Teste de Aceitação:** Garantir que a API está pronta para ser consumida por uma aplicação front-end ou por outros serviços, atendendo a todos os critérios de aceite.

2.4. Tipos de Teste

- **Teste Funcional:**
    - **Objetivo:** Garantir que todas as funcionalidades listadas no escopo operem conforme o esperado.
    - **Técnica:** Execução de casos de teste para cada endpoint, utilizando dados válidos e inválidos para verificar o comportamento da API em diferentes cenários.
    - **Critérios de Aceite:** A API deve retornar os resultados esperados (dados e códigos de status) para cada requisição.
- **Teste de Validação de Schema:**
    - **Objetivo:** Garantir que as respostas da API estejam em conformidade com os schemas definidos no arquivo `api_docs.json`.
    - **Técnica:** Utilizar ferramentas que possam validar automaticamente o corpo da resposta contra o schema OpenAPI.

3. Ferramenta de Teste

Para a execução dos testes manuais e automatizados da API, será utilizada a seguinte ferramenta:

- **Postman:** Para criar e executar as requisições HTTP para cada endpoint. Será feita a criação de coleções de testes, a verificação de respostas (códigos de status, corpo, cabeçalhos) e a automação de validações básicas usando scripts.

Caso haja tempo, os testes automatizados poderão ser feitos utilizando o Rest Assured do Java.

4. Regras de Negócio

As regras de negócio da API são definidas pela especificação no arquivo `api_docs.json`. Os principais pontos a serem validados nessa versão inicial incluem:

- A criação de um pedido deve estar associada a um fornecedor e a um ou mais itens existentes.
- Não deve ser possível deletar um fornecedor que esteja associado a um pedido existente.
- A atualização de um item em estoque deve refletir corretamente a quantidade total.
- Validações de campos obrigatórios e formatos de dados (ex: `integer`, `string`, `date-time`) devem ser aplicadas em todas as requisições `POST` e `PUT`.

5. Configuração do Ambiente

- **Descrição da Máquina de Teste:**
    - **Hardware:** 16GB de RAM, processador i9 11800h, 500g de armazenamento livre.
    - **Software:**
        - Sistema Operacional Linux
        - Postman instalado.
        - Acesso à rede para consumir a API.
- **Descrição do Ambiente da API:**
    - **URL Base:** `http://localhost:8080`
    - **Banco de Dados:** Postgres local rodando via Docker.

6. Cronograma de Teste

- **Duração Total Estimada:** 3 dias
    - **Dia 1:** Planejamento, criação dos casos de teste e configuração do ambiente no Postman/Insomnia.
    - **Dia 2:** Execução dos testes funcionais (caminhos felizes e alternativos).
    - **Dia 3:** Execução de testes negativos, testes exploratórios e elaboração do relatório final de teste.

7. Abordagem dos Testes

A metodologia utilizada será baseada em uma abordagem Ágil.

- **Processo de Teste:**
    1. **Revisão da Documentação:** Análise detalhada do `api_docs.json`.
    2. **Criação dos Casos de Teste:** Elaboração dos casos de teste para cada endpoint.
    3. **Configuração do Ambiente:** Preparação da coleção no Postman/Insomnia com todas as requisições.
    4. **Execução dos Testes:** Execução planejada dos testes, incluindo testes exploratórios para encontrar falhas não previstas.
    5. **Registro de Defeitos:** Documentação de todos os bugs encontrados.
    6. **Relatório Final:** Elaboração de um relatório com os resultados dos testes.

8. Casos de Teste

Os casos de teste serão detalhados em um documento que está em desenvolvimento pois os requisitos ainda não estão totalmente fechados

9. Tabela de Risco

Abaixo estão alguns dos riscos identificados, com base na criticidade da funcionalidade.

Categoria	Risco	Impacto	Probabilidade	Severidade (Impacto * Probabilidade)
Pedidos	Falha ao salvar um pedido com itens inválidos	5	3	15
Estoque	Falha na atualização da quantidade de um item no estoque após um pedido	5	4	20
Fornecedores	Permitir cadastro de fornecedores com CNPJ duplicado	4	3	12
Medicamentos	Permitir cadastro de medicamento sem lote ou validade	4	3	12
Geral	API não retorna erro 404 para recursos não encontrados	3	4	12
Geral	API vulnerável a ataques de Injeção (se houver consultas diretas)	5	2	10
Geral	Falha de autorização/autenticação (não especificado, mas um risco comum)	5	2	10

10. Formato de Registros de Defeitos e Ferramentas

- **Ferramenta para Registro de Defeitos:**
  - Quadro do Jira
- **Formato do Registro:**
  - **ID do Defeito:** Identificador único.
  - **Título:** Resumo claro e conciso do problema.
  - **Endpoint:** Qual endpoint da API 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 resposta da API (corpo, status, etc.).
  - **Severidade:** (Crítica, Alta, Média, Baixa).

11. Comunicação e Relatórios

- **Ferramentas de Comunicação:**

    - **Whatsapp e Discord:** Para comunicação rápida e esclarecimento de dúvidas.
- **Relatórios:**
    - Ao final do ciclo de testes, será gerado um **Relatório Final de Testes** em formato de documento (Google Docs), contendo:
        - Resumo dos testes executados.
        - Número de casos de teste executados, aprovados e reprovados.
        - Lista de defeitos encontrados (com suas severidades).
        - Recomendação final sobre a qualidade da API (se está pronta para produção ou não).

Futuramente será implementado uma esteira de testes que gera um relatório do Allure Reports para melhor visualização