Relatório Geral dos Testes

(Gerencia de Configuração e Mudanças)

FarmacoCheck

Data de conclusão do relatório: 13/01/2025

Autores: Silvio Martins Santos.

Repositório: https://github.com/silviosmsantos/FarmacoCheck

1. INTRODUÇÃO

Neste relatório, será descrito o processo de testes, uma das etapas do processo de verificação e validação do produto FarmacoCheck, realizados até o momento. Trata-se de uma aplicação desenvolvida para a disciplina de Gerência de Configuração e Mudanças, voltada à checagem de interação entre medicamentos, que busca facilitar o ato de conferir se existem interações entre dois medicamentos que possam prejudicar pacientes. Um ponto importante a ser mencionado é que os testes foram realizados de acordo com os recursos disponíveis e com o grau de prioridade classificado no documento de requisitos. A escala de prioridade varia de 1 a 5, sendo 1 a maior prioridade e 5 a menor. Ao longo deste relatório, serão abordados os testes realizados (unidade, integração e sistema), as ferramentas utilizadas e os principais problemas identificados.

- **Observações:** Detalhes sobre como verificar a cobertura dos testes estão no arquivo *README.md*, disponível no repositório.
    - Após inicial a aplicação e acessar a rota http://127.0.0.1:8000/uml é possível visualizar um diagrama de classes gerado pelo próprio sistema.

2. FERRAMENTAS UTILIZADAS

Aqui consta a lista de ferramentas empregadas para a realização dos testes:

- **Pest**[1]: ferramenta empregada para a execução de testes unitários e de integração. Além disso, foi utilizado para medir a cobertura dos testes, indicando a porcentagem do código que foi exercitado pelos teste;

[1] https://pestphp.com/

- **K6**[2]: utilizado para testes de carga/estresse do sistema em ambiente de produção;
- **Git Actions**[3]**:** ferramenta de integração contínua utilizada para automatizar o processo de testes e deploy (CI/CD);

3. TESTES DE UNIDADE

3.1. Objetivo

Os testes de unidade foram aplicados para garantir o funcionamento esperado de componentes isolados da aplicação. Para este projeto, cada método de classe foi considerado como uma unidade. Todos os testes de unidade realizados estão disponíveis em:

A. farmaco-check-app\tests\Unit

3.2. Casos de teste

As seguintes classes foram testadas, os teste das classes abaixo podem ser conferidos nos diretórios mencionados anteriormente (A) :

farmaco-check-app\tests\Unit	Tests class
Interaction.php	InteractionsModelTest.php
Medicine.php	MedicinesModelTest.php
	MedicinesInteractionsTest.php
User.php	UserTest.php

3.3. Resultados

A cobertura de testes não atingiu o esperado para as classes listadas acima. Alguns métodos não foram exercitados como esperado corretamente por falta de planejamento para construção de um banco de dados para testes. Porém foi possível verificar e validar que os métodos de cada classe funcionam como esperado. Outros métodos e outras classes não tiveram condições de serem testados pelo fator tempo.

4. TESTES DE INTEGRAÇÃO

[2] https://k6.io/open-source/
[3] https://github.com/features/actions

4.1. Objetivo

Os testes de integração realizados durante o desenvolvimento do FarmacoCheck visam verificar o comportamento esperado da aplicação quando diferentes módulos interagem entre si. Os arquivos que constam dos testes de integração podem ser acessados pelo seguinte diretório:

B. farmaco-check-app\tests\Feature

4.2. Casos de teste

Os seguintes arquivos são compostos por testes de integração:

- ProfileTest.php;
- AuthenticationTest.php;
- EmailVerificationTest.php
- PasswordConfirmationTest.php
- PasswordResetTest.php
- PasswordUpdateTest.php;
- RegistrationTest.php

4.3. Resultado

A cobertura de testes de integração não atingiu o esperado quando analisada de forma geral para a aplicação. Algumas classes e suas interações não foram exercitadas corretamente devido à falta de um planejamento adequado para a construção de um banco de dados de testes. No entanto, com base nas funcionalidades testadas, foi possível verificar e validar que as classes e seus métodos, especialmente os relacionados à autenticação de usuários, funcionam conforme o esperado. Outros métodos e classes não foram testados devido à limitação de tempo.

5. TESTES DE SISTEMA

5.1. TESTES FUNCIONAIS EXPLORATÓRIOS

Os testes funcionais exploratórios foram realizados conforme o incremento de cada funcionalidade, permitindo uma navegação livre pela aplicação para identificar comportamentos inesperados ou problemas não capturados pelos testes automatizados. Esses testes buscam avaliar cenários reais de uso, sendo realizados também em ambiente de produção. As áreas cobertas pelos testes incluem:

1. **Checagem de permissão/acesso de usuários**: Verificação de que cada tipo de usuário tem acesso apenas às suas funcionalidades específicas, conforme descrito no documento de requisitos.
2. **Gerenciamento de usuários**: Verificação de todas as funcionalidades relacionadas ao gerenciamento de usuários por um SuperAdmins.
3. **Gerenciamento de informações sobre medicamentos e interações**: Avaliação das funcionalidades que envolvem exibição e manipulação de informações sobre medicamentos e suas interações.
4. **Busca de interações**: Avaliação do funcionamento da funcionalidade principal da aplicação, que permite buscar interações entre medicamentos e exibir as informações relevantes ao usuário.

5.2.1 Resultados

Os testes exploratórios ajudaram a identificar problemas menores na interface de usuário, como falhas na exibição de mensagens longas, formulários relacionados a número de telefone, e problemas de navegação em dispositivos móveis, que não foram capturados pelos testes automatizados. Os maiores achados com os testes exploratórios podem ser conferidos na Seção 6.

5.2. TESTES DE CARGA/ESTRESSE

Os testes de carga e estresse foram realizados para avaliar um dos requisitos não funcionais da aplicação, especificamente sua capacidade de lidar com um grande número de usuários e requisições simultâneas, simulando cenários de alta demanda. Esses testes mediram o desempenho do sistema em ambiente de produção sob condições extremas e o comportamento da aplicação sob estresse prolongado.

- **Ferramenta utilizada**: K6, essa ferramenta utilizada para simular múltiplos acessos simultâneos e monitorar o comportamento da aplicação.

- **Cenários testados:**
    - **Carga**: simulação de 100 usuários simultâneos navegando por um catálogo de produtos, onde o tempo de resposta esperado era ser menor ou igual a 3 segundos por requisição..

- **Estresse:** simulação de um aumento progressivo no número de usuários até 400 simultâneos interagindo com uma página de catálogo. O tempo de resposta das requisições foi monitorado durante esse aumento chegando ao tempo de 22 segundos para aplicação responder uma requisição.

5.3.1. Resultados:

A aplicação se comportou como esperado com até 100 usuários simultâneos, atendendo às requisições dentro do tempo de resposta esperado de 3 segundos, o que atende ao requisito não funcional correspondente (RNF003). O sistema também apresentou dificuldades em processar um grande número de mensagens simultâneas, sugerindo a necessidade de otimização no módulo de comunicação. Além disso, os testes revelaram gargalos de desempenho e na renderização de página, indicando que são necessárias melhorias para suportar cargas mais elevadas. Todo o processo realizado com o testes de carga e estresse podem ser verificados no documento no repositório:

- **tests_docs/teste_carga_estresse-K6.pdf**

6. PRINCIPAIS PROBLEMAS ENCONTRADOS

Esta seção apresenta os principais problemas identificados por meio dos testes realizados até a data de elaboração deste relatório. Todos os problemas relatados foram descobertos utilizando os recursos humanos e de tempo disponíveis. Cada problema já foi reportado e no momento há um entendimento claro sobre suas causas e possíveis soluções.

6.1. PROBLEMA 1: Cache

- **Descrição:** Ao navegar por diferentes rotas da aplicação, é necessário clicar mais de uma vez para que a ação de navegação aconteça. Além disso, alguns componentes, como imagens, demoram para carregar na primeira vez.
- **Verificação**: Este problema pode ser verificado apenas acessando a aplicação por alguns minutos e acessando diferentes rotas.
- **Possíveis Soluções:**
    - Implementar melhorias no gerenciamento de cache, garantindo que os dados sejam carregados de maneira eficiente.
    - Otimizar a carga de recursos estáticos, como imagens, utilizando técnicas como lazy loading ou compressão de imagens para reduzir o tempo de carregamento.

- Verificar as configurações de cache no servidor e no navegador para garantir que as rotas e os componentes sejam devidamente armazenados em cache, evitando a necessidade de múltiplos cliques.

6.2. PROBLEMA 2: NÃO SOBREPOSIÇÃO DE FEEDBACK E PERMANÊNCIA DE MENSAGENS

- **Descrição**: Ao realizar uma consulta entre interações em que há uma interação válida, uma mensagem de sucesso é exibida. No entanto, essa mensagem não desaparece, e, caso seja realizada outra consulta (onde não há interação), a nova mensagem aparece sobre a anterior, sem sobrepô-la ou removê-la após um tempo determinado.
- **Verificação**: O problema pode ser verificado acessando a aba de consultas e realizando uma consulta válida (que retorna uma interação) e, em seguida, uma consulta não válida (que não retorna uma interação), sem clicar no botão de "resetar consulta".
- **Possíveis soluções**:
  - Implementar um sistema de sobreposição de mensagens, de forma que a mensagem mais recente substitua a anterior.
  - Adicionar um temporizador para que as mensagens desapareçam automaticamente após um período predefinido.
  - Garantir que, ao realizar uma nova consulta, qualquer mensagem anterior seja removida antes de exibir a nova.

6.3. PROBLEMA 3: VARÁVEL DE AMBIENTE DE DEBUG ESTÁ ATIVA EM PRODUÇÃO

- **Descrição:** Quando ocorre um comportamento inesperado no sistema devido a falhas de permissão ou erros no banco de dados, a tela de debug do Laravel é exibida ao usuário, o que não é adequado para um ambiente de produção..
- **Verificação**: O problema pode ser observado quando o usuário encontrar algum *bug* ou o banco de dados apresenta falha principalmente.
- **Possíveis soluções**:
  - Alterar a variável de ambiente APP_DEBUG de true para false no ambiente de produção para desabilitar a exibição de erros e evitar a exposição de informações sensíveis.

- Criar uma página de erro personalizada para ambientes de produção, em caso de falha, o usuário receba uma mensagem amigável sem expor dados técnicos.
- Implementar monitoramento de erros para que, em caso de falha, os desenvolvedores sejam notificados sem expor o erro aos usuários finais.

7. CONCLUSÃO

O processo de testes, da aplicação foi realizado com sucesso até o presente momento, utilizando testes de unidade, integração e sistema, respeitando os recursos disponíveis. Embora a aplicação esteja, em partes, operando conforme o esperado, foram identificados alguns problemas preocupantes. As soluções propostas são essenciais para melhorar a estabilidade e a experiência do usuário. A implementação dessas melhorias será crucial para corrigir as falhas observadas em busca de garantir a qualidade do produto final. Além disso, a continuidade do processo de avaliação da aplicação é importante para assegurar a sua evolução constante.