Projeto Final
Verificação, Validação e Testes de Software
Instituto de Computação, Unicamp

Prof. Dra. Eliane Martins
Eduardo Boccato Pires de Camargo 194286
Giovanne Lucas Dias Pereira Mariano 173317
Natan Beltrão da Cunha Pevidor Carvalho 184972

13 de agosto de 2022

Resumo

No projeto final, com o objetivo de exercitar as técnicas de verificação, validação e testes de software, foi analisada a aplicação shopizer, uma solução open-source para o problema tradicional do e-commerce.

No contexto de testes de caixa preta, foram utilizadas técnicas de testes baseados em classe de equivalência e valores-limite.

Por outro lado, no contexto de testes de caixa branca, foram testados os critérios de cobertura de estados e cobertura de arestas para os modelos de estados desenvolvidos sobre a documentação da aplicação de acordo com as histórias de usuário fornecidas na atividade 5.

Os resultados indicaram um grande número de falhas na API da aplicação. Algumas falhas estão relacionadas à inconsistência no comportamento da API e algumas falhas estão relacionadas com a falta de validação do back-end que depende exclusivamente do front-end para evitar problemas.

O repositório com o código desenvolvido para o projeto pode ser encontrado no github em giolucasd/shopizer-tests.

1 Introdução

O projeto final tem por objetivo colocar em prática as técnicas de verificação, validação e testagem de software estudadas ao longo da disciplinas, tendo como sistema objeto das atividades o software shopizer **[Q1]**.

A shopizer é uma solução open-source para e-commerce com a pretensão de ser facilmente adaptável às diferentes demandas que administradores de sistemas e-commerce podem requirir.

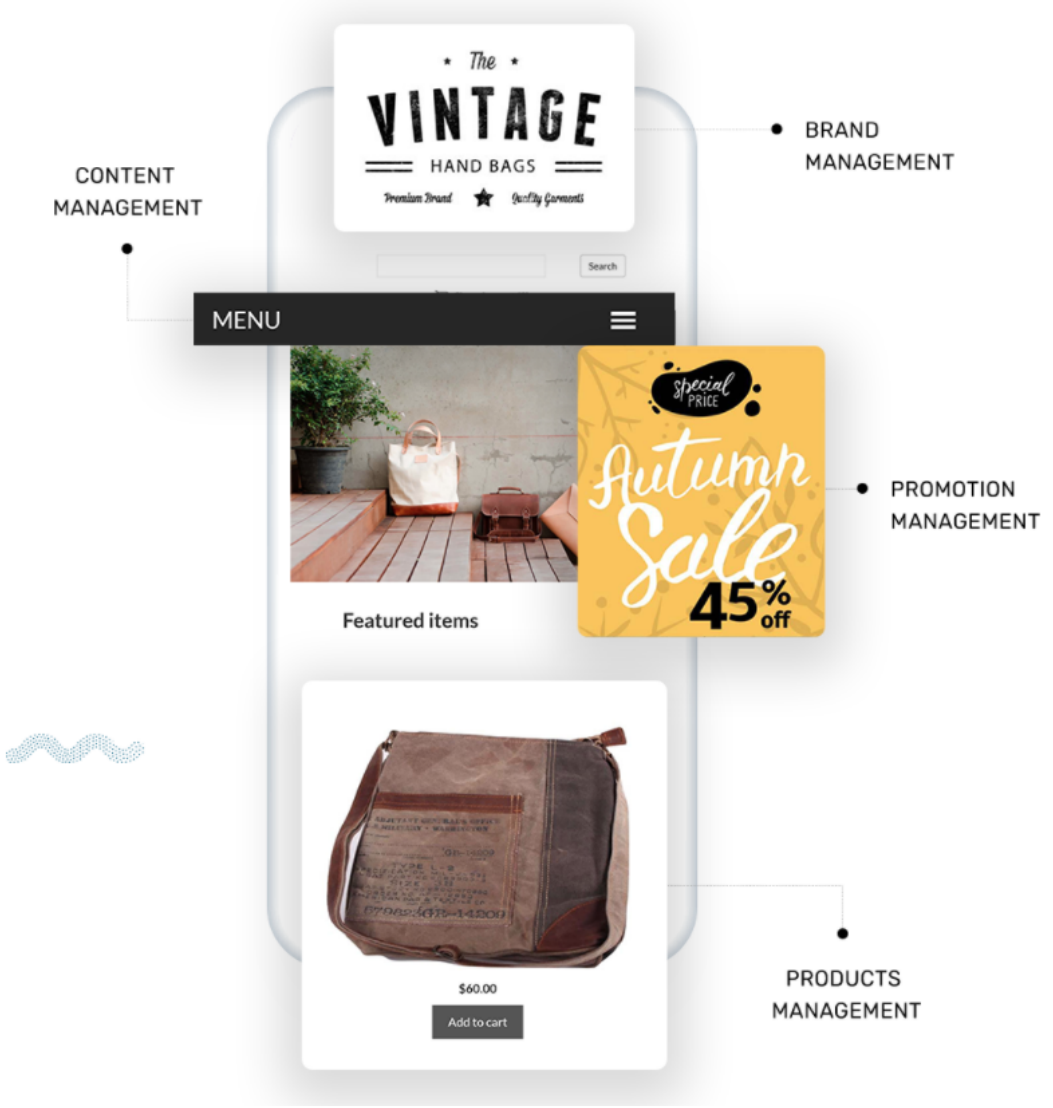

Figura 1: Arte de divulgação das principais funcionalidades providas pela shopizer.

O principal focos dos testes é a API do sistema que contém uma documentação swagger com as definições técnicas.

As respostas das perguntas do relatório final serão identificadas no texto com um indicador em negrito, como, por exemplo **[Q1]** indicaria que o texto contém a resposta da pergunta 1.

2 Metodologia

Para exercitar os conceitos de testes baseados em classes de equivalência, bem como testes baseados em valores-limite, foram exercitados os endpoints de CRUD (criação, leitura, atualização e remoção) de cliente da API. Os endpoints de gerenciamento de cliente foram escolhidos como foco inicial dos testes porque tiveram o maior índice na matriz de capacitações do sistema, de acordo com a documentação.

Neste contexto, foram utilizados os seguintes critérios:

- Para cada variável de entrada mapeado, foram testados um valor inválido e um valor válido. Tendo em vista que os domínios são strings e valores de múltipla escolha, todas as variáveis possuem apenas uma classe inválida e uma classe válida.

- Para exercitar os critérios de valores-limite, foram testados os valores de tamanho mínimo e máximo válidos e valores inválidos com tamanho mínimo - 1 e tamanho máximo + 1.

Ou seja, com as entradas devidamente mapeadas, os casos de teste gerados com tais critérios se basearam, seguindo os critérios acima **[Q2]**, nos seguintes valores limites para as variáveis de entrada:

Valores Limite		
Variável de entrada	Tipo	Tamanho máximo
Address	Varchar	256
City	Varchar	100
Postal Code	Varchar	20
State Province	Varchar	100
Country	Int	ID
Zone	Big Int	ID
First Name	Varchar	64
Last Name	Varchar	64
Phone	Varchar	32
Email Address	Varchar	96

Tabela 1: Tabela com os valores limites para as variáveis de entrada.

Dada essa abordagem, foram gerados 44 casos de teste para a operação de criação, 4 para a de leitura, 46 para a operação de atualização e 4 para a operação de deleção **[Q3]**.

Para exercitar as técnicas de testagem baseadas em modelos de estados, por outro lado, foram desenvolvidos testes com valores de entrada interessantes para testar os fluxos de interação entre APIs mais interessantes de acordo com algum critério de cobertura (cobertura de estados ou cobertura de arestas).

Para os casos de uso descritos, foram gerados os modelos de estados das figuras 2, 3, 4 e 5.

Com estes modelos, utilizando o software GraphWalker, foi possível simular a execução de modo que foram obtidos os grafos das figuras 6, 7, 8 e 9.

Utilizando o software Graph Coverage, foi possível gerar casos de teste para cobrir os critérios de cobertura selecionados.

H1. Como usuário/a, eu quero me cadastrar para que eu possa acompanhar o andamento dos meus pedidos e receber notificações sobre promoções.

- O usuário fornece as informações requeridas pela API correspondente.
- Caso o usuário já esteja cadastrado, o sistema responde com a resposta adequada.
- O usuário entra na sua área, onde poderá: alterar sua senha, verificar se os dados de sua conta foram criados corretamente.
- O usuário pede a lista de pedidos recentes. Caso a lista não esteja vazia, o usuário deve receber seu conteúdo.
- O usuário sai do aplicativo dando um logout.

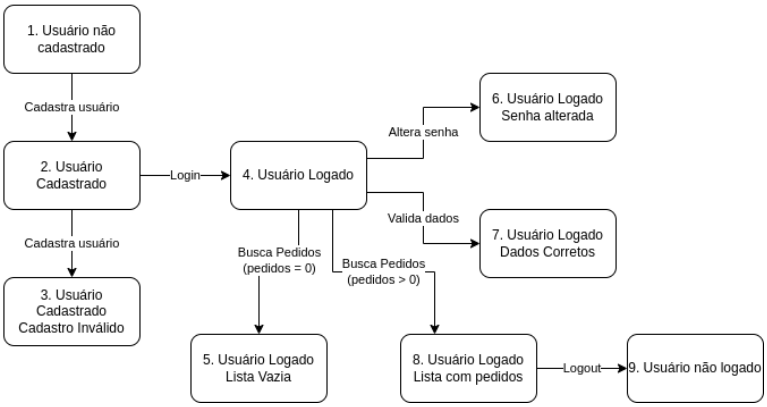

Figura 2: Modelo de estados para a história de usuário H1.

H2. Como usuário/a já cadastrado, eu quero obter a variação de preços de um produto específico (ex. geladeiras) eu possa escolher o que seja mais barato

- O usuário deve se autenticar no sistema. Caso a autenticação seja inválida, deve tentar novamente ou então se cadastrar.
- Uma vez autenticado, o usuário pede a lista de preços de um determinado produto.
- Caso o produto indicado não exista, o usuário deve ser informado.
- Caso não haja variantes de preços para o produto, o sistema informa ao usuário.
- O usuário sai do aplicativo.

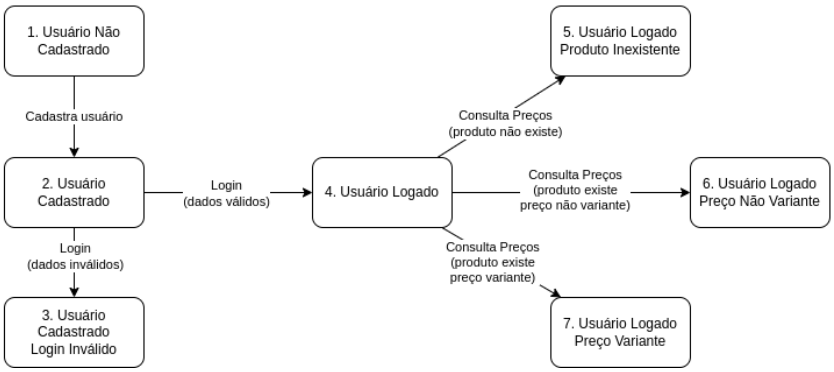

Figura 3: Modelo de estados para a história de usuário H2.

Para cada história de usuário, foi encontrado um conjunto de teste que cobre todos os critérios desejados de acordo com a tabela 2.

História	Conjunto de testes
H1	[2,4,5] [2,4,6] [2,4,7] [2,4,8,9] [1,2,3]
H2	[2,4,7] [2,4,5] [2,4,6] [1,2,3]
H3	[1,2] [2,5,2] [2,5,6,2] [2,3,5,2] [2,4,5,2] [2,3,4,5,2]
H4	[1,2] [2,6,2] [2,6,10,2] [2,5,4,2] [2,6,8,9,2] [2,6,8,7,2] [2,3,6,2] [2,5,6,2] [2,6,5,4,2] [2,3,5,4,2]

Tabela 2: Tabela com os conjuntos de teste para cada história de usuário.

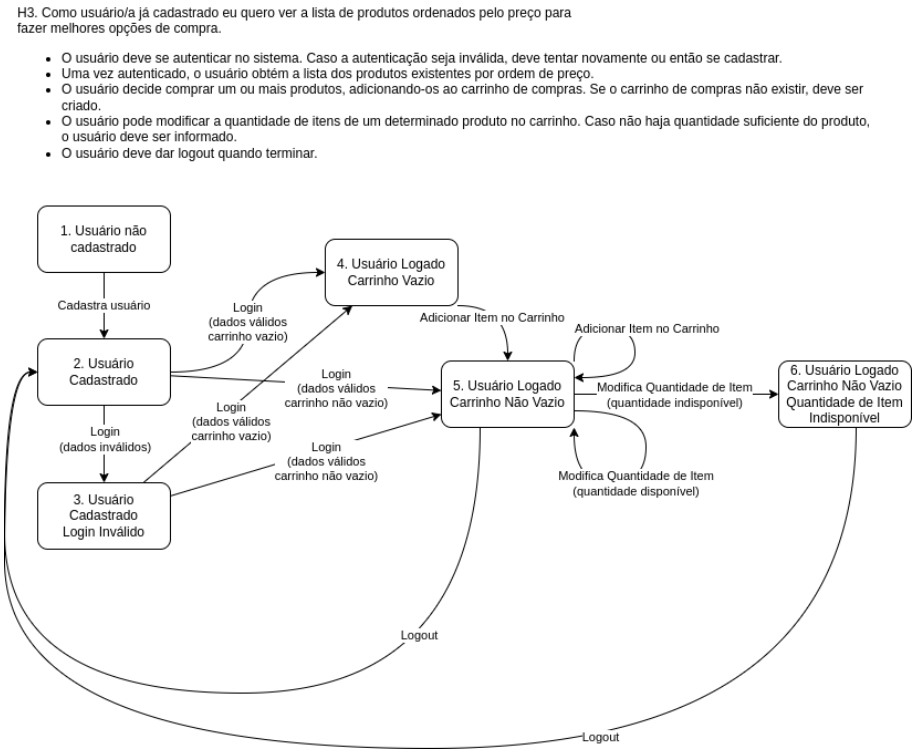

Figura 4: Modelo de estados para a história de usuário H3.

É interessante notar que em todas as histórias, os casos de teste associados a cobertura de arestas cobriu, também, os demais critérios de cobertura.

3 Resultados

Para os testes de caixa preta, baseados em valores limite, foi encontrado o seguinte sumário dos resultados obtidos:

Endpoint	Nº erros status	Nº erros match	Nº erros crash	Nº testes bem-sucedidos
Create	13	0	11	20
Read	0	0	0	4
Update	15	0	10	21
Delete	0	0	0	4

Tabela 3: Tabela com o sumário dos resultados obtidos nos testes de classe de equivalência e valores limite **[Q5]**.

Não foram encontrados erros de match, pois não foram testados os valores retornados, apenas os resultados esperados e a existência dos recursos. Ao total, foram encontradas 49 falhas e 49 sucessos nos testes **[Q4]**.

Estes números mostram, entretanto, que existem dois grandes problemas com a API de CRUD de clientes da shopizer:

- Falta de validação dos dados recebidos: a API não valida os dados, apenas

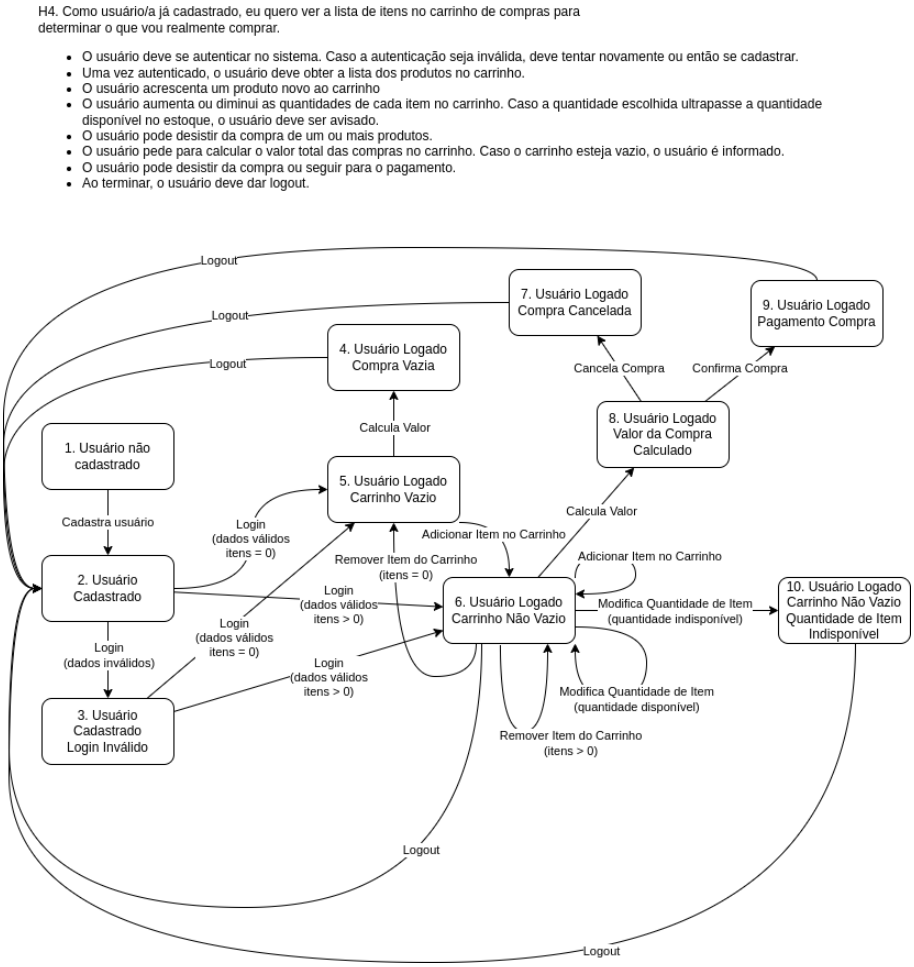

Figura 5: Modelo de estados para a história de usuário H4.

tenta salvá-los no banco de dados diretamente, confiando cegamente na validação do front-end, uma prática extremamente perigosa e susceptível a ataques.

- Inconsistência na padronização dos status de retorno: além de não seguir a convenção proposta na documentação swagger, a API tem diversas inconsistências geradas pelas diferenças de modelagem no banco de dados entre os campos. Este problema ocorre, pois, uma vez que não existe validação dos campos, diferentes modelagens dos campos geram diferentes erros ou comportamentos inesperados, gerando grande inconsistência no comportamento da API.

4 Discussão

Com o auxílio das aulas e exercícios teóricos fornecidos ao longo da disciplina, o projeto dos casos de teste de acordo com os critérios estabelecidos em cada uma das técnicas de testagem se provou bastante intuitivo.

Por outro lado, a utilização do Gherkin para gerar os casos de teste, apesar de interessante do ponto de vista funcional e do laboratório tutorial, se mostrou uma grande dificuldade para a implementação de alguns casos de teste, uma vez que não

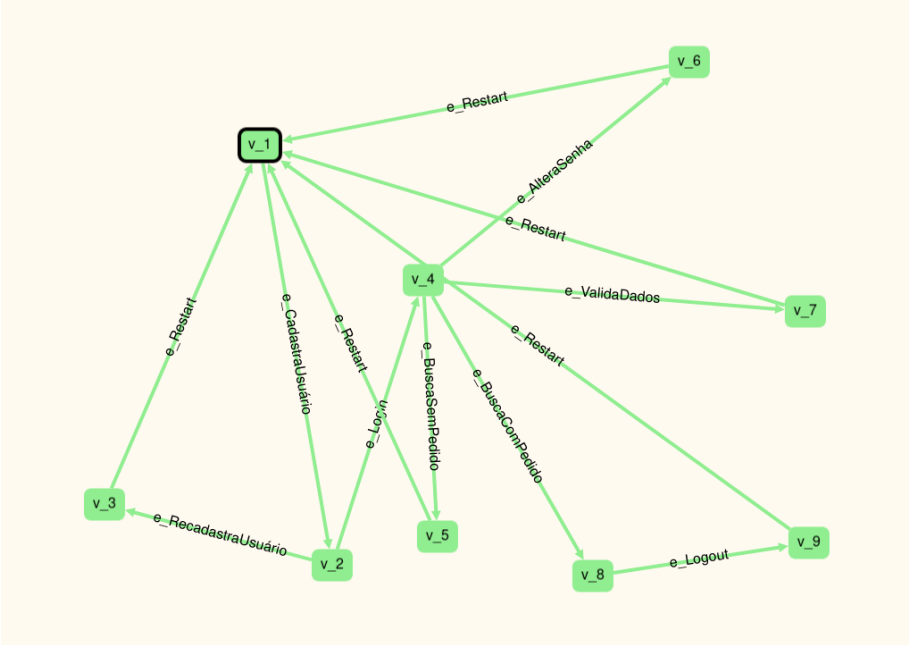

Figura 6: Execução do modelo de estados para a história de usuário H1 no software GraphWalker.

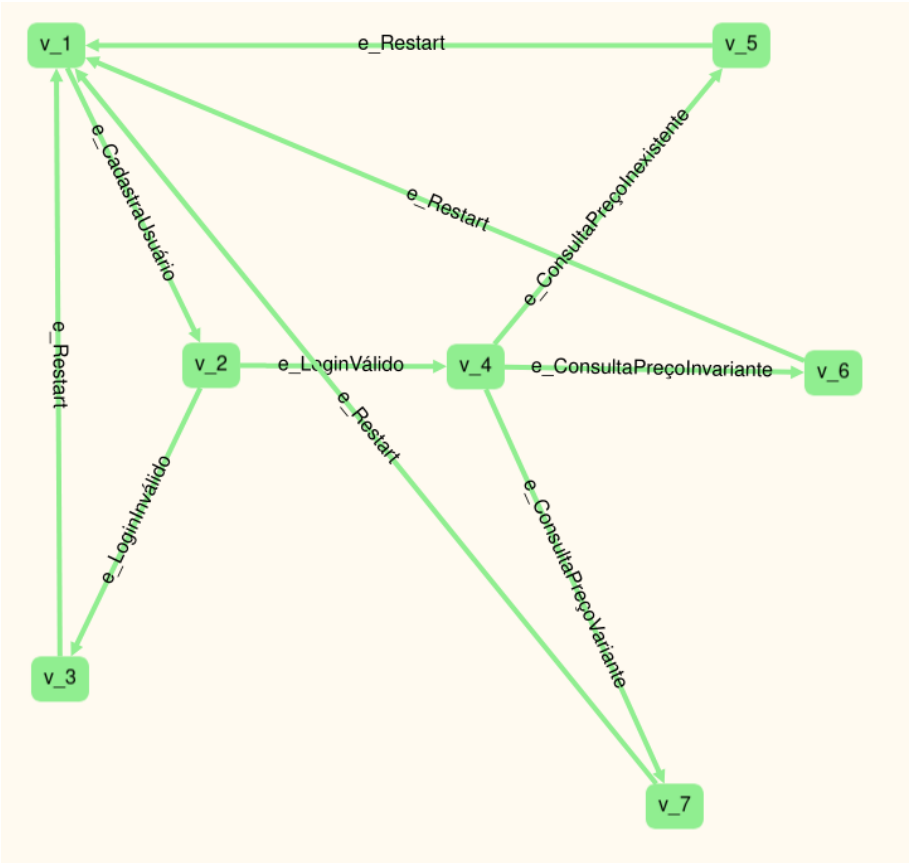

Figura 7: Execução do modelo de estados para a história de usuário H2 no software GraphWalker.

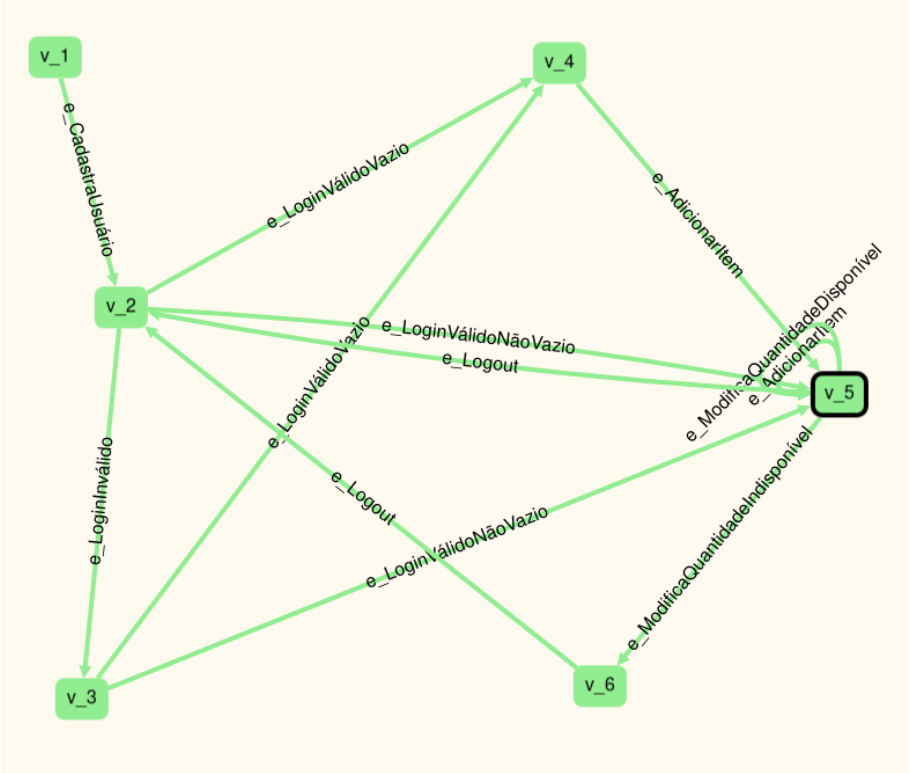

Figura 8: Execução do modelo de estados para a história de usuário H3 no software GraphWalker.

conseguimos construir mocks de dados para executar testes de leitura, atualização e deleção de registros. Para contornar esta dificuldade utilizamos o endpoint de criação de registro para criar os dados. Esta prática faz com que os testes gerados percam o caráter unitário, uma vez que problemas na API de criação podem interferir no resultado dos demais testes.

Outra dificuldade encontrada foi o uso da aplicação em teste (shopizer) nos laboratórios do IC, uma vez que rodar o contêiner para executá-la exigiu desenvolvermos algumas gambiarras junto ao suporte técnico **[Q6]**.

Uma recomendação interessante para as próximas equipes de teste seria não perder tempo com pequenos detalhes **[Q7]** ou se apegar a problemas como a necessidade de criar registros usando a API de criação nos demais testes. Começar pelo que é fácil de fazer e então trabalhar iterativamente para aumentar a cobertura dos testes é uma ótima estratégia e vai garantir que, se existirem problemas que podem ser rapidamente identificados, então eles serão identificados rapidamente.

5 Conclusão

Os experimentos com testes de API servem como uma forte base para validar o entendimento de que usar estratégias de testagem como a utilização de classes de equivalência, valores-limite e critérios de cobertura para fluxo de dados ou controle é muito eficiente para identificar a maioria das falhas mais comuns em softwares.

As técnicas de testagem, além de direcionarem o plano de testes e tornarem o processo objetivo, se mostraram bastante eficazes para proporcionar testes limpos e

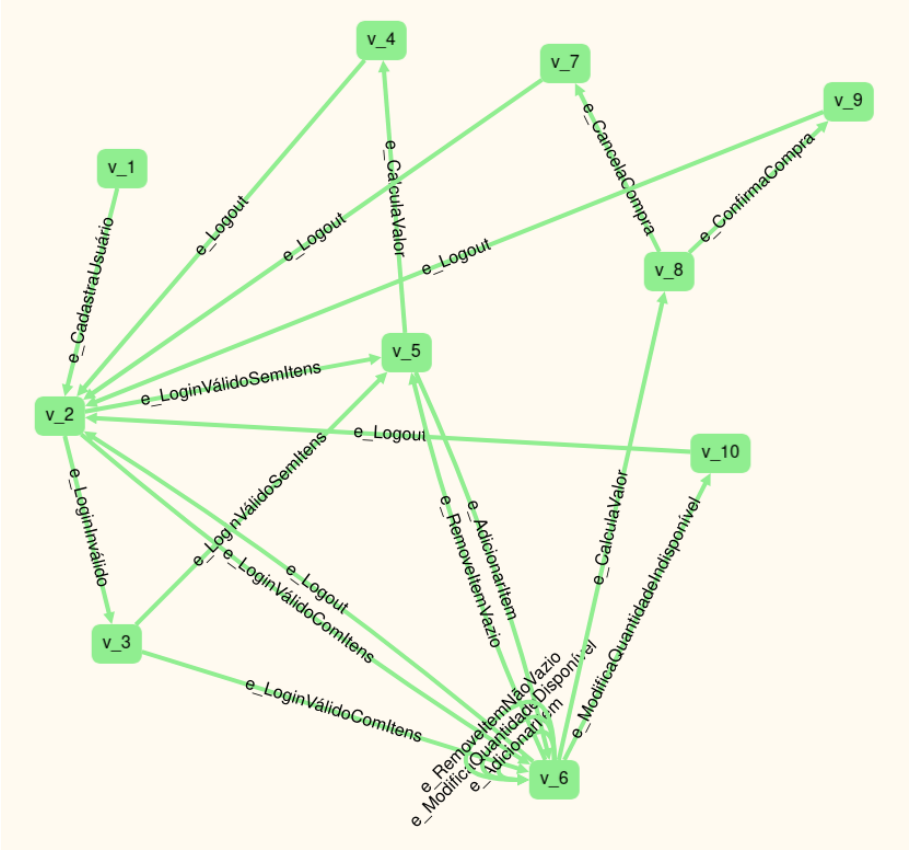

Figura 9: Execução do modelo de estados para a história de usuário H4 no software GraphWalker.

legíveis. Neste contexto, a utilização do Gherkin também facilita bastante a construção de um conjunto de testes agnóstico à linguagem de implementação e acessível a diferentes grupos de desenvolvedores ou, até mesmo, construtores de requisitos envolvidos no projeto e implementação do software.