

**Plano de testes da**

**API REST Serverest**



Escrito por:

Leonardo Da Silva Oliveira

Sumário:

[**Introdução:** 3](#_Toc113261932)

[**Termos e acrônimos:** 3](#_Toc113261933)

[**Objetivos:** 4](#_Toc113261934)

[**Escopo:** 4](#_Toc113261935)

[**Mapa mental:** 5](#_Toc113261936)

[**Suíte de casos de teste:** 6](#_Toc113261937)

[**Sobre os testes de fluxo:** 6](#_Toc113261938)

[**Sobre os testes de regressão:** 14](#_Toc113261939)

[**Estado inicial dos testes:** 15](#_Toc113261940)

[**Estratégia de teste:** 17](#_Toc113261941)

[**Priorização de teste:** 17](#_Toc113261942)

[**Candidatos para automação:** 17](#_Toc113261943)

[**Ferramentas:** 17](#_Toc113261944)

[**Divulgação dos resultados:** 18](#_Toc113261945)

[**Observações complementares:** 18](#_Toc113261946)

# **Introdução:**

Este documento apresenta um plano de testes para verificar a usabilidade e funcionamento da API do Serverest, suas funcionalidades e verbos disponibilizados para uso de lojistas e usuários e suas respostas dada a sua documentação oficial.

Tendo isso em vista a Serverest é uma API REST gratuita que simula uma loja virtual com intuito de servir de material de estudos de testes de API, ela possui armazenamento próprio de estoque, cadastro de usuários, administradores e carrinhos de compras. No site oficial e na documentação da API embutida nos arquivos do github possuem detalhamento das possíveis respostas HTTP que a API pode retornar e as mensagens que as acompanham embutidas nos arquivos JSON.

## **Termos e acrônimos:**

|  |  |
| --- | --- |
| Termo | Descrição |
| **API** | “Application Programming Interface” que significa em tradução para o português “Interface de Programação de Aplicativos”. |
| **Float** | É um tipo de variável com valor fracionado numeral. |
| **GitHub** | É um serviço online baseado no sistema de controle de versão Git. Ele permite que os desenvolvedores colaborem e façam mudanças em projetos compartilhados enquanto mantêm um registro detalhado do progresso. |
| **HTTP** | HTTP é um protocolo de transferência que possibilita que as pessoas que inserem a URL do seu site na Web possam ver os conteúdos e dados que nele existem. A sigla vem do inglês “Hypertext Transfer Protocol”. |
| **Input** | Entrada de dados ou de objeto, tradução livre: “Entrada”. |
| **Int** | É um tipo de variável com valor inteiro numeral. |
| **JSON** | Em computação, JSON, um acrônimo de JavaScript Object Notation, é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas. |
| **REST** | API REST, também chamada de API RESTful, é umainterface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST. |
| **Scripts** | Scripts é um texto com uma série de instruções e ou comados escritos para serem seguidos. |
| **String** | Strings são sequências imutáveis de caracteres. |

## **Objetivos:**

Este plano de testes tem os seguintes objetivos:

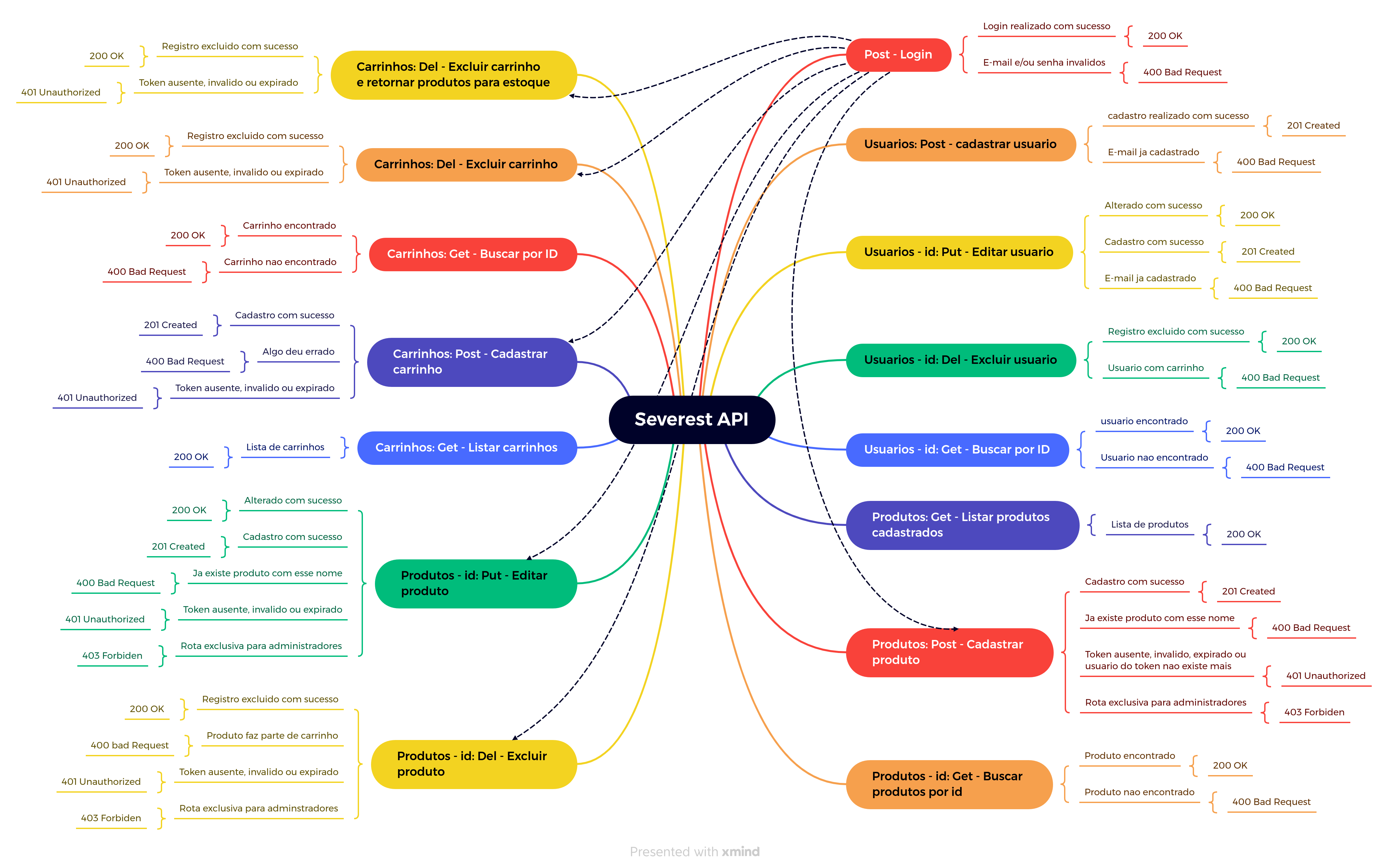
* Levantar informações pertinentes quanto ao funcionamento correto dos verbos da API
* Rever as respostas recebidas e comparar com as respostas descritas na documentação
* Descrever e priorizar o conjunto de requisitos a serem testados
* Determinar os tipos de testes e as ferramentas que serão utilizadas
* Definir a infraestrutura necessária para os testes
* Determinar como os resultados serão divulgados

# **Escopo:**

Neste documento será abordado um plano de testes para autenticar os fluxos encontrados de lojistas e usuários da API Serverest, o qual é utilizado para criar/modificar/deletar itens da loja, criar/modificar/deletar/logar usuários/administradores, criar/deletar carrinhos, remover ou devolução de itens da loja. Assim como testes regressivos dos verbos da API, onde serão testados diversos tipos de input diferentes, String, Int, Float, valores negativos e zerados ou nenhuma entrada.

Alguns verbos da API funcionam por base de uma autentificação via Bearer token onde vários verbos apenas precisam desse header para sua função viável. “Bearer authentication” pode ser entendido como “give access to the bearer of this token” em tradução livre “Dê acesso ao portador desse token” o token bearer é uma string criptografada, e esse token é utilizado para verificação de autorização administrativa da API, e acompanhamento das compras de usuários até a sua conclusão, seja de compra com retirada dos itens do estoque ou de desistência, com a devolução dos itens ao estoque, ou para acesso das funções da loja, como adicionar/remover/modificar itens.

# **Mapa mental:**

[[1]](#footnote-1)Figura 1: Mapa mental completo de acordo com a documentação

De acordo com a documentação oficial da API, esses são os verbos que podem ser utilizados, eles estão divididos em três pastas principais, Usuários, Produtos e carrinhos. Um único verbo fica fora dessa padronização que é o verbo POST login, que é essencial para o funcionamento correto de diversos verbos da API pois gera o token bearer já mencionado, e que é utilizado na API (Ele é mostrado em destaque pelas várias setas que dele saem, mostrando em quais verbos ele se mostra necessário).

# **Suíte de casos de teste:**

Os testes deverão ser organizados em duas pastas distintas, uma sendo para os testes de fluxo, imitando interações que normalmente lojistas ou usuários teriam com a API, e a outra para testes de regressão onde são testadas todas as possibilidades de entrada de dados para os verbos disponibilizados.

Importante ressaltar que os testes não tem nenhum intuito de testar capacidades limites de usuários, quantidade máxima de produtos e a sua consequente na performance na API, por isso durante os testes quando usuários e itens da loja forem criados com o intuito de teste seja de um fluxo ou de regressão esse usuário e/ou item serão deletados da API.

Os verbos testados devem ser organizados seguindo uma lógica e identificados por uma ID com a seguinte regra:

AZ00-00

* Teste de fluxo ou regressivo (F ou R)
* Inicial da pasta principal
* Posição numeral da pasta
* Número da posição do verbo

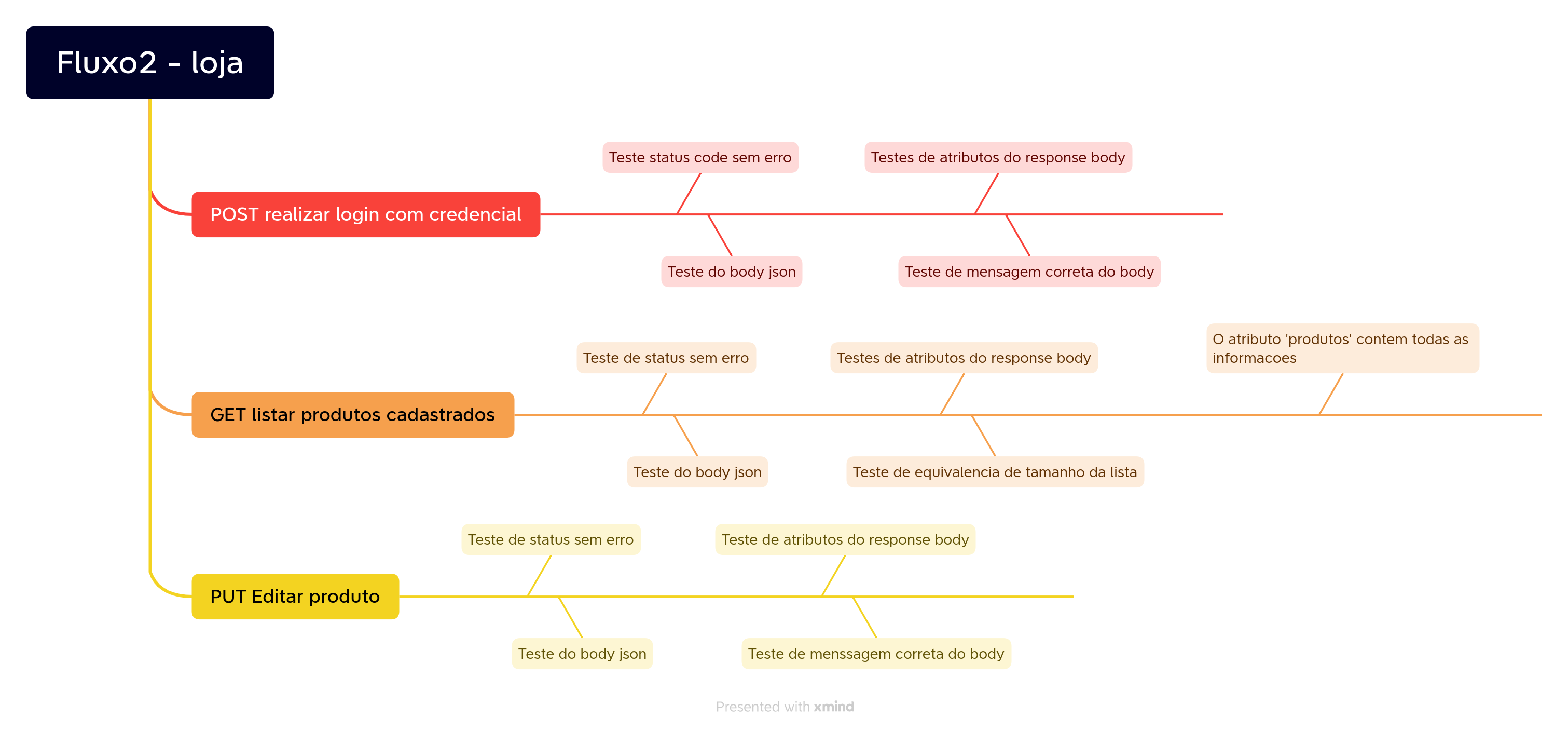
## **Sobre os testes de fluxo:**

Os testes devem ter dois fluxos de lojistas, um com testes de adicionar editar e remover produtos da loja, e um segundo com fluxo incremento ou decréscimo de estoque.

ID = FL01

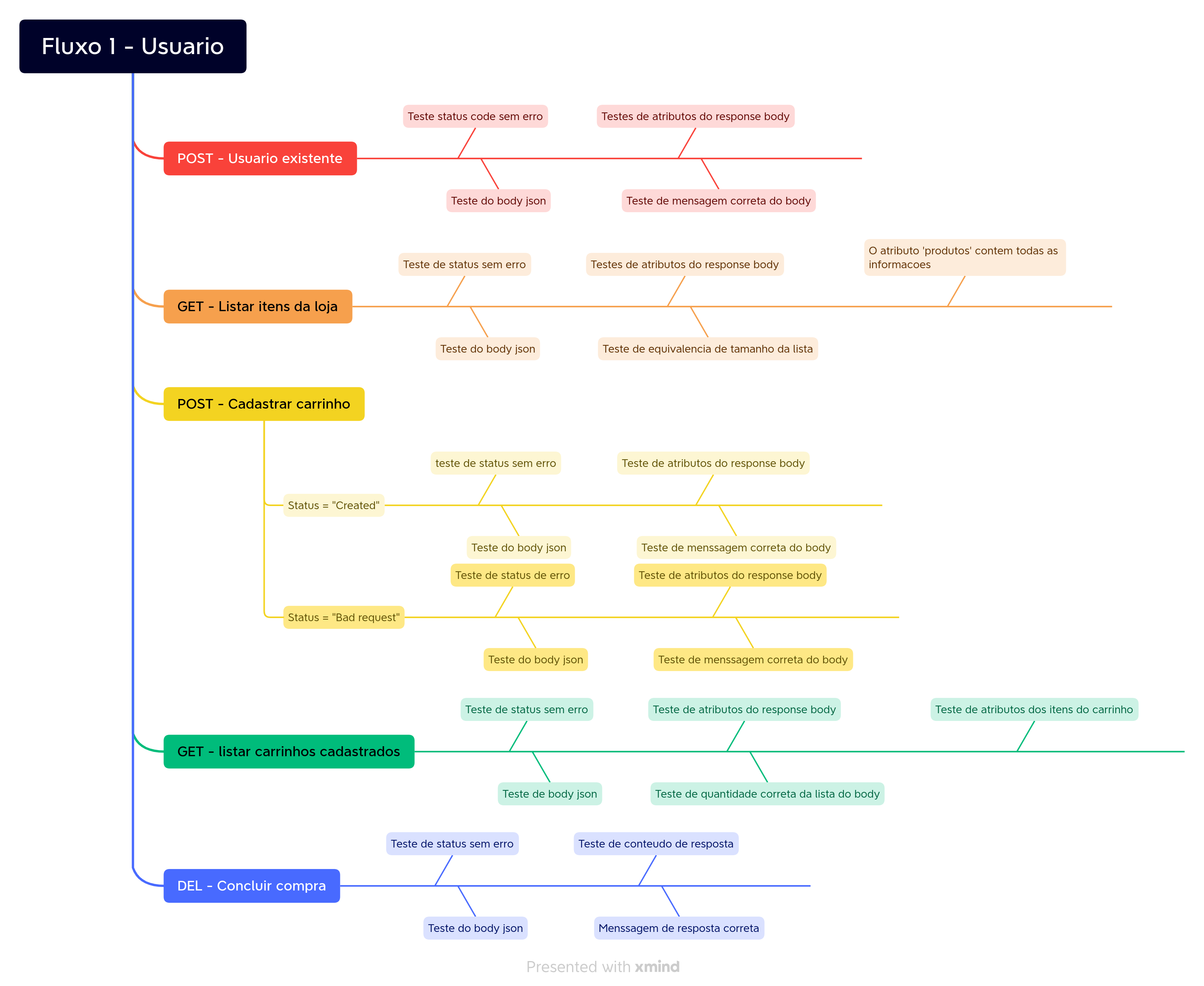
[[2]](#footnote-2)Figura 2: Mapa mental do fluxo de testes de loja número 1, e os testes presentes no fluxo.

ID = FL02

[[3]](#footnote-3)Figura 3: Mapa mental do fluxo de testes de loja número 1, e os testes presentes no fluxo.

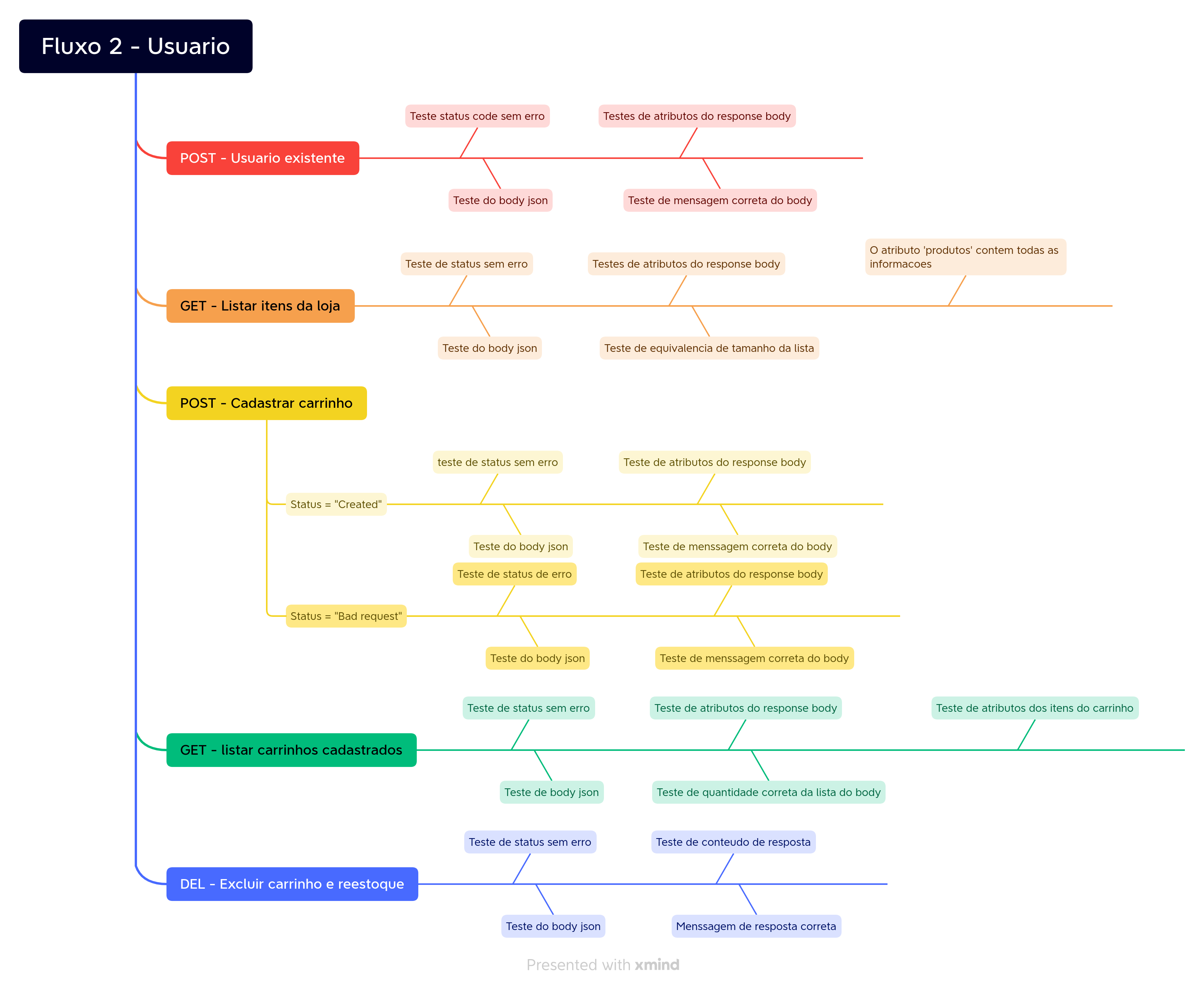
Os fluxos de usuários serão 4, o primeiro sendo de um usuário já cadastrado fazendo login, vendo os itens da loja, escolhendo um cadastrando um carrinho com número x de itens, visualizar o carrinho e concluir a compra.

ID = FU01

[[4]](#footnote-4)Figura 4: Mapa mental do fluxo de testes de usuários número 1, e os testes presentes no fluxo.

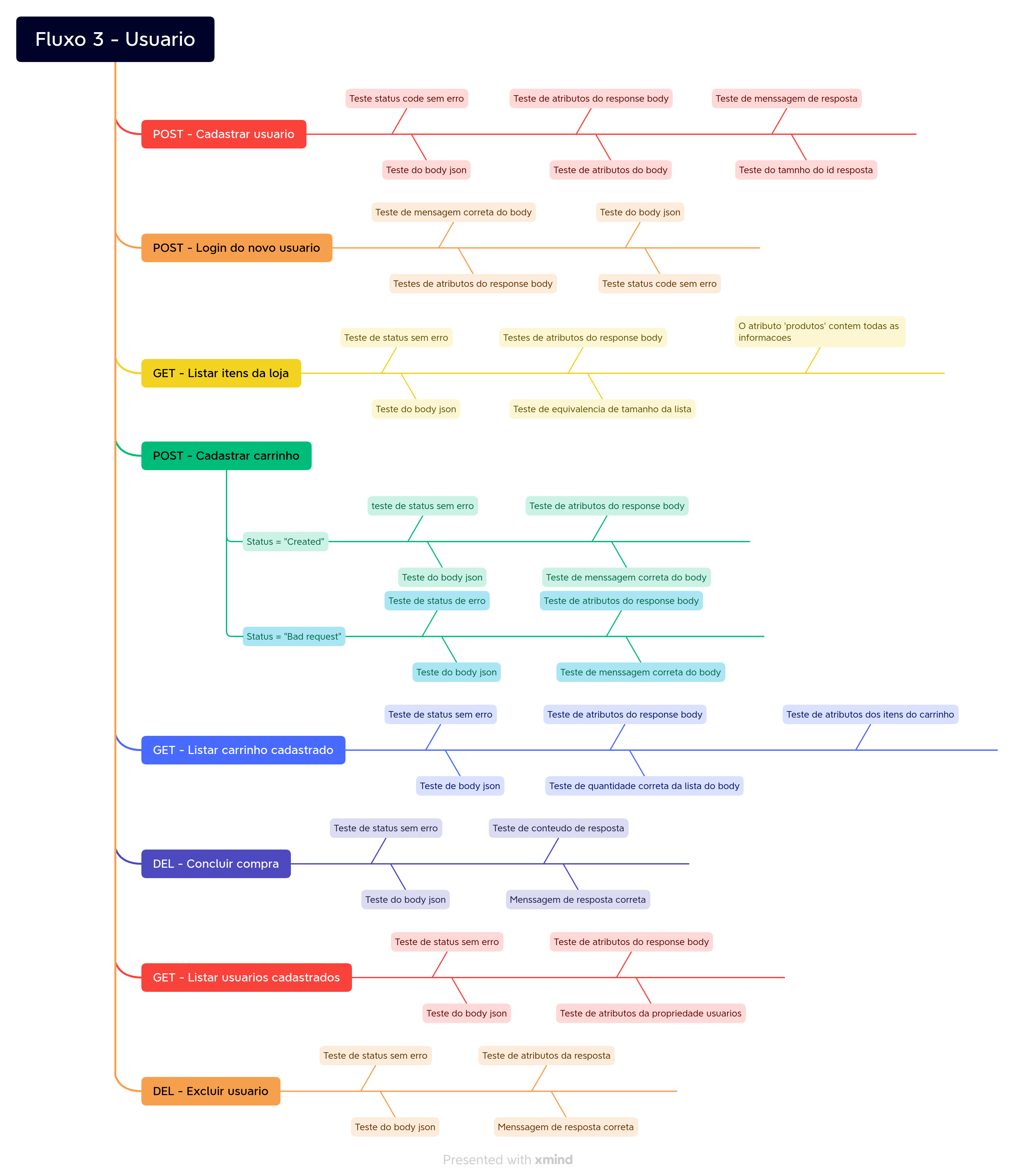
O segundo será de um usuário já cadastrado fazendo login, vendo os itens da loja, escolhendo um cadastrando um carrinho com número x de itens, visualizar o carrinho e desistindo da compra ao final.

ID = FU02

[[5]](#footnote-5)Figura 5: Mapa mental do fluxo de testes de usuários número 2, e os testes presentes no fluxo.

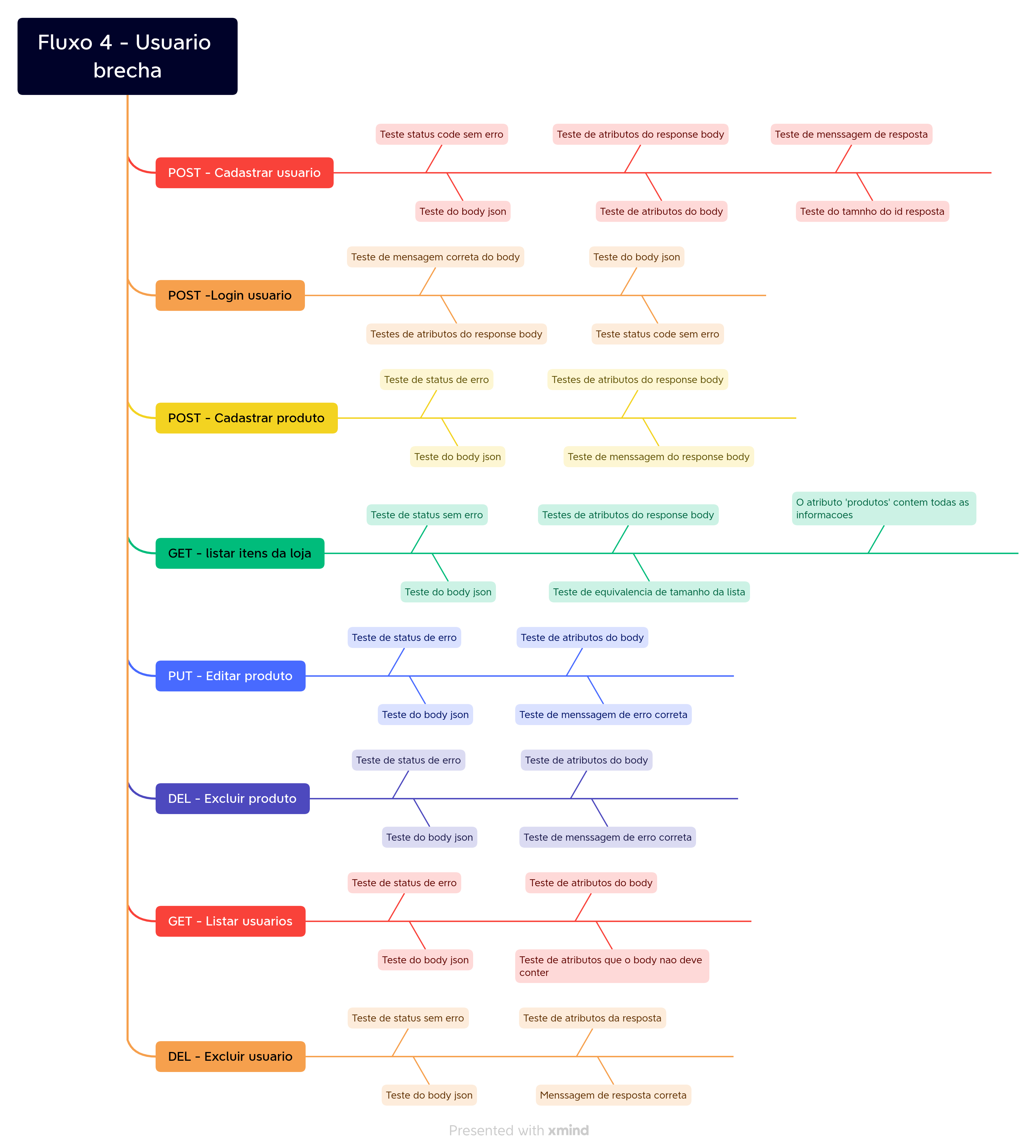
O terceiro fluxo de usuário será de um usuário não cadastrado que cria uma nova conta na API, faz o login com essa conta, visualiza os itens da loja, escolhe um item cadastra um carrinho, visualiza o carrinho e conclui a compra.

ID = FU03

[[6]](#footnote-6)Figura 6: Mapa mental do fluxo de testes de usuários número 3, e os testes presentes no fluxo.

O quarto e último fluxo de usuários testa vulnerabilidades da API quando um usuário não administrador tenta utilizar verbos da API de modo a modificar/adicionar/deletar itens da loja.

ID =FU04

[[7]](#footnote-7)Figura 7: Mapa mental do fluxo de testes de usuários número 4, e os testes presentes no fluxo.

## **Sobre os testes de regressão:**

Os testes de regressão estão organizados em pastas como na documentação, usuários, produtos e carrinhos. Os testes de cada verbo dessas categorias se encontram dentro de pastas com as mesmas nomenclaturas, assim como subpastas. O nome de cada pasta final indica qual verbo estará sendo testado.

Começando com a pasta **usuarios**, subpasta cadastro, são testadas diferentes formas de se abordar o cadastramento de um usuário qualquer.

Pasta cadastro auto:

Subpasta "email-attack", ID = RU01: Gera-se e-mails aleatórios, com nomes aleatórios. São utilizados caracteres especiais nos locais de nome e e-mail, há a probabilidade de o nome e/ou o e-mail ir em branco, essa probabilidade é aumentada com o número maior de campos vazios na lista geradora de nomes e e-mails aleatórios. A formação de e-mail final é testada com a combinação do nome mais a finalização de e-mail a partir de uma terceira lista onde a capacidade de reconhecimento de e-mail da API é testada.

Subpasta “psswd-attack”, ID = RU02: São testadas diferentes formas de se cadastrar um usuário com e-mail válido, mas senhas “inválidas”. RU02-01 tenta cadastrar com string vazia sem espaço; RU02-02 tenta cadastrar com string vazia com espaço; RU02-04 tenta cadastrar com numerais sem ser string; RU02-05 tenta cadastrar com uma sequência de caracteres aleatória sem aspas no body.

Subpasta “wrong-adm”, ID = RU03: São testadas diferentes entradas de body na verificação de cadastro de usuário administrador. RU03-01 testa entrada booleana ao invés de string; RU03-02 testa entrada numeral.

Saindo da pasta cadastro as seguintes subpastas da pasta **usuarios** são:

Pasta id-Buscar-usuario, ID = RU04:

RU04-01 testa o verbo, com o header, mas com o header “id” vazio; RU04-02 sem a opção de header “accept-aplicatio/json” desmarcada e header “aleatório”. RU04-03 testa o verbo para que ele liste todos os usuarios sem token bearer e verifica se não são passados dados sensiveis.

Pasta id-excluir-usuario, ID = RU05:

RU05-01 testa o verbo com o header “id” vazio; RU05-02 Testa o verbo com o header “id” com uma string aleatoria.

Pasta id-editar-usuario-auto, ID = RU06:

RU06-04 testa a capacidade de uma conta sem poderes administrativos alterar outra conta no sistema. RU06-05 testa a capacidade de uma conta se dar o poder administrativo. RU06-07 testa a modificação de uma conta sem nenhum token bearer presente. RU06-08 testa o verbo com token bearer expirado e tenta modificar uma conta inexistente no sistema.

Seguindo para a próxima pasta a pasta de **produtos-auto** ID = RP01:

RP01-02 testa uma tentativa de cadastro de produto somente com nome e com os campos preco e quantidade em valores Int, e descricao vazia; RP01-03 testa uma tentativa de cadastro somente com nome e com os campos preco e quantidade em valores Int, e descricao string; RP01-04 testa a tentativa de cadastro de produto com nome e descricao string e preco com Int com 16 casas decimais e quantidade Int 0; RP01-05 testa a tentativa de cadastro de um produto com nome e descricao string, preco Int 0 e descricao com valor Int com 16 casas decimais; RP01-06 testa a tentativa de cadastro com nome e descricao string e preco negativo e quantidade 0; RP01-07 testa a tentativa de cadastro de produto com nome e descricao string e preco float e quantidade 0; RP01-08 testa a tentativa de cadastro de produto com nome e descricao string e preco Int positivo e quantidade 0; RP01-09 tenta modificar um produto com nome e descricao string para ter preco negativo e quantidade 0; RP01-10 tenta modificar um produto com nome e descricao string para ter preco float e quantidade 0.

A próxima pasta é a de **carrinhos-auto** ID = RC01:

RC01-02 testa todos os atributos da resposta para cada carrinho; RC01-03 testa a criação de carrinho com id de produto inexistente; RC01-04 testa com id correto a criação de carrinho com quantidade negativa: RC01-05 testa a criação de carrinho com id de produto existente e quantidade 0; RC01-06 testa a criação de carrinho com id de produto existente e quantidade string; RC01-08 testa a deleção de carrinho sem header “Accept-aplication/json”; RC01-10 testa a deleção e retorno de estoque sem header “accept-aplication/json”.

A última pasta é a pasta do verbo principal **login** ID = RL01:

RL01-01 tenta fazer login com senha vazia; RL01-02 tenta fazer login com email vazio e senha string; RL01-03 tenta fazer login com email e senha vazios; RL01-04 tenta fazer login com email correto e senha Int ao invez de string; RL01-05 tenta fazer login de um usuário existente com uma capitalização de uma letra do email; RL01-06 tenta fazer login de um usuario existente sem os headers “content-Type aplication/json” e “Accept aplication/json”.

## **Estado inicial dos testes:**

Para a realização dos testes o servidor deverá estar configurado da seguinte forma:

Modificação desses nomes e id’s necessitara de reconfiguração do ambiente de testes.

**Produtos:**

|  |
| --- |
| {  "quantidade": 3,  "produtos": [  {  "nome": "bandana do Icarly",  "preco": 690,  "descricao": "item editado",  "quantidade": 51998990,  "\_id": "BeeJh5lz3k6kSIzA"  },  {  "nome": "Notebook lenovo ryzen5 3500u - 8gb 1tb",  "preco": 925,  "descricao": "notebook",  "quantidade": "14410",  "\_id": "K6leHdftCeOJj8BJ"  },  {  "nome": "Pedra filosófal",  "preco": 488,  "descricao": "Mouse",  "quantidade": "327891",  "\_id": "yeDdGmRmqodGzXns"  }  ]  } |

**Usuarios cadastrados:**

|  |
| --- |
| {  "quantidade": 3,  "usuarios": [  {  "nome": "Fulano da Silva",  "email": "cicrano@qa.com.br",  "password": "teste",  "administrador": "true",  "\_id": "2Huny5EhV94f9bH8"  },  {  "nome": "leonardo Silva",  "email": "leonardo@test.com",  "password": "12345",  "administrador": "true",  "\_id": "cFUgrMaMwU8eyxiN"  },  {  "nome": "Editado funcional",  "email": "edicaodeesquema@testao.yt",  "password": "teste",  "administrador": "true",  "\_id": "vlHskDNeKH9qSXGV"  }  ]  } |

**Carrinho cadastrados:**

|  |
| --- |
| {  "quantidade": 1,  "carrinhos": [  {  "produtos": [  {  "idProduto": "BeeJh5lz3k6kSIzA",  "quantidade": 2,  "precoUnitario": 470  },  {  "idProduto": "K6leHdftCeOJj8BJ",  "quantidade": 1,  "precoUnitario": 5240  }  ],  "precoTotal": 6180,  "quantidadeTotal": 3,  "idUsuario": "oUb7aGkMtSEPf6BZ",  "\_id": "qbMqntef4iTOwWfg"  }  ]  } |

# **Estratégia de teste:**

A estratégia dos testes é testar a maior quantidade diferentes de entradas que os verbos permitem passar e que deveriam ou não passar, trocando entradas de string para Int e float ou strings sem aspas como é pedido por alguns verbos da API. Também será testado a capacidade de modificação do banco de dados por parte de usuários não administradores.

# **Priorização de teste:**

A priorização de testes será da seguinte forma:

1. Prioridade para funcionalidade da API para lojas e usuários
2. Prioridade para a correta delimitação de poder de usuários dentro da API
3. Prioridade para a segurança de dados sensíveis
4. Prioridade para entradas indesejadas de carrinhos
5. Prioridade para cadastramento indevido de produtos
6. Prioridade para login correto
7. Prioridade para entrada correta de email e senha no cadastro

# **Candidatos para automação:**

Os candidatos para a automação de testes da API são todos os testes de fluxo e alguns testes de regressão, os testes de regressão marcados como possíveis candidatos a automação estarão indicados com a nomenclatura “auto” no nome da pasta.

# **Ferramentas:**

As ferramentas necessárias para a realização dos testes são:

Computador (de preferência do ano de 2018 ou mais recente)

Postman e/ou Newman (para a realização dos testes)

Nodejs e NPM (para rodar o servidor localmente)

Microsoft 365 (Para escrever os documentos)

Xmind (Para criar os mapas mentais)

# **Divulgação dos resultados:**

Os resultados dos testes serão postos no mesmo repositório criado para o plano de testes, com o nome de arquivo sendo: “analise\_de\_testes.docx”. Ele deverá ser entregue em conjunto com o plano de testes e os arquivos necessários para a reprodutibilidade dos testes em outras máquinas.

# **Observações complementares:**

É possível encontrar as imagens dos mapas mentais em maior qualidade na pasta Xmind do repositório do plano de testes da API Serverest.

Agradeço a todos do time Rocketman 04 da compasso, alguns dos testes foram pensados após o levantamento e possibilidade de interação errônea da API.

1. Figura: Mapa mental completo de acordo com a documentação [↑](#footnote-ref-1)
2. Figura 2: Mapa mental do fluxo de testes de loja número 1. [↑](#footnote-ref-2)
3. Figura 3: Mapa mental do fluxo de testes de loja número 1. [↑](#footnote-ref-3)
4. Figura 4: Mapa mental do fluxo de testes de usuários número 1, e os testes presentes no fluxo. [↑](#footnote-ref-4)
5. Figura 5: Mapa mental do fluxo de testes de usuários número 2, e os testes presentes no fluxo. [↑](#footnote-ref-5)
6. [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)