Plano de teste de API

1 - Apresentação

Este documento constitui o plano de teste para a API ServerRest. Seu propósito fundamental é delinear e comunicar a estratégia, o escopo, os recursos necessários e o cronograma para todas as atividades de teste relacionadas à API.

2. Objetivo Estratégico dos Testes

- **Validação da Conformidade REST:** Assegurar que a API adere estritamente aos princípios da arquitetura REST. Isso envolve a verificação do uso correto dos verbos HTTP (GET, POST, PUT, DELETE) para as operações de Criar, Ler, Atualizar e Excluir (CRUD) , a utilização de substantivos no plural para a nomeação de recursos nos endpoints e o emprego adequado dos códigos de status HTTP para comunicar de forma clara e padronizada os resultados das operações.
- **Verificação das Regras de Negócio:** Garantir que a lógica de negócio que simula a operação de uma loja virtual seja implementada de forma correta e consistente. O foco principal será na validação de fluxos críticos, como a autenticação de usuários, o gerenciamento de perfis (administrador vs. cliente), o controle de inventário de produtos e o ciclo de vida completo de um carrinho de compras, desde sua criação até a conclusão ou cancelamento da compra.
- **Garantia de Integridade e Segurança:** Validar a persistência e a integridade dos dados manipulados através dos endpoints da API. Isso inclui verificar se as operações de criação, atualização e exclusão se refletem corretamente no estado do sistema. Adicionalmente, os testes devem validar rigorosamente os mecanismos de autenticação e autorização, garantindo que apenas usuários autenticados e com as permissões adequadas (e.g., administradores) possam realizar operações restritas.
- **Identificação e Relatório de Defeitos:** Identificar, documentar e reportar de forma sistemática quaisquer desvios do comportamento esperado. Cada defeito encontrado será reportado com informações claras e acionáveis, incluindo passos para reprodução, dados de entrada utilizados e resultados obtidos versus esperados, a fim de facilitar o processo de depuração e correção pela equipe de desenvolvimento.

3. Escopo e Delimitação dos Testes

Recursos em Escopo

O escopo dos testes abrangerá a totalidade dos recursos (endpoints) expostos pela API ServeRest, conforme sua documentação oficial. Os principais recursos a serem testados são:

- `/login`: Endpoint responsável pela autenticação de usuários e geração de token de autorização.
- `/usuarios`: Conjunto de endpoints para operações CRUD relacionadas ao gerenciamento de usuários.
- `/produtos`: Conjunto de endpoints para operações CRUD relacionadas ao gerenciamento de produtos no catálogo da loja.
- `/carrinhos`: Conjunto de endpoints para o gerenciamento de carrinhos de compras, incluindo a adição de produtos e a conclusão ou cancelamento de uma compra.

Funcionalidades em Escopo

As seguintes funcionalidades e tipos de teste serão executados:

- **Testes Funcionais:** Validação de todos os verbos HTTP (GET, POST, PUT, DELETE) para cada endpoint em escopo.
- **Validação de Contrato:** Verificação da conformidade das respostas da API com os schemas JSON definidos na documentação, garantindo a consistência do contrato da API.
- **Autenticação e Autorização:** Teste dos fluxos de acesso, incluindo requisições com e sem token de autenticação no cabeçalho, bem como a validação de perfis de usuário (administrador vs. cliente).
- **Tratamento de Erros:** Validação sistemática das respostas de erro, incluindo a verificação dos códigos de status HTTP (séries 4xx e 5xx) e a clareza das mensagens de erro retornadas.

4. Análise da Aplicação e Regras de Negócio

A API ServeRest simula uma arquitetura de e-commerce, onde diferentes recursos são interdependentes para realizar um fluxo de negócio completo. O fluxo principal de um usuário envolve autenticação (`/login`), navegação pelo catálogo de produtos (`/produtos`), adição de itens a um carrinho de compras (`/carrinhos`) e, finalmente, a conclusão da compra.

A interdependência entre os recursos `/usuarios` , `/produtos` e `/carrinhos` é o ponto central da lógica de negócio. Testar cada endpoint de forma isolada é insuficiente para garantir a qualidade do sistema como um todo. A regra de negócio mais complexa, a conclusão de uma compra, envolve a leitura de dados de `/carrinhos` e `/produtos` , a aplicação de uma lógica de

verificação de estoque e, subsequentemente, a atualização do estoque em `/produtos`. Uma falha em qualquer um desses passos invalida todo o fluxo de negócio.

Regras de Negócio por Recurso

- **Autenticação (`/login`):**
  - Um usuário com credenciais válidas (e-mail e senha) deve receber um token de autorização (`Bearer token`).
  - O envio de credenciais inválidas deve resultar em uma resposta de erro de autenticação com status `401 Unauthorized`.
- **Usuários (`/usuarios`):**
  - O cadastro de novos usuários é permitido, mas o campo `email` deve ser único. Uma tentativa de cadastrar um e-mail já existente deve retornar um erro com status `400 Bad Request`.
  - Os usuários são diferenciados pela propriedade booleana `administrador`.
  - Apenas usuários com perfil de administrador podem cadastrar outros usuários como administradores.
  - A exclusão de um usuário deve remover permanentemente seus dados do sistema.
- **Produtos (`/produtos`):**
  - Apenas usuários autenticados com perfil de administrador podem cadastrar, editar ou excluir produtos.
  - Tentativas de realizar essas operações por usuários não-administradores ou não autenticados devem ser bloqueadas com status `403 Forbidden` ou `401 Unauthorized`, respectivamente.
  - Não é permitido cadastrar um produto com um nome que já exista no sistema.
  - Cada produto possui um campo `quantidade` que representa o estoque disponível.
- **Carrinhos (`/carrinhos`):**
  - Qualquer usuário autenticado pode criar um carrinho de compras.
  - Ao concluir uma compra (`DELETE /carrinhos/concluir-compra`), o sistema deve validar se a quantidade de cada produto no carrinho está disponível no estoque (`/produtos`).
  - Se o estoque for suficiente, a quantidade em estoque dos produtos deve ser decrementada, e o carrinho do usuário deve ser "limpo" ou marcado como concluído.

- Se o estoque for insuficiente para qualquer um dos produtos, a operação deve falhar, o estoque não deve ser alterado, e uma mensagem de erro apropriada deve ser retornada.
- Ao cancelar uma compra (`DELETE /carrinhos/cancelar-compra`), o estoque não deve ser alterado, e o carrinho do usuário deve ser liberado.

Histórias de Usuário (User Stories)

US 001 - [API] Usuários

Sendo um vendedor de uma loja Gostaria de poder me cadastrar no Marketplace do ServeRest Para poder realizar as vendas dos meus produtos

DoR (Definition of Ready)

- Banco de dados e infraestrutura para desenvolvimento disponibilizados;
- Ambiente de testes disponibilizado.

DoD (Definition of Done)

- CRUD de cadastro de vendedores (usuários) implementado (CRIAR, ATUALIZAR, LISTAR E DELETAR);
- Análise de testes cobrindo todos verbos;
- Matriz de rastreabilidade atualizada;
- Automação de testes baseado na análise realizada;

Acceptance Criteria (Critérios de Aceite)

- Os vendedores (usuários) deverão possuir os campos NOME, E-MAIL, PASSWORD e ADMINISTRADOR;
- Não deverá ser possível fazer ações e chamadas para usuários inexistentes;
- Não deve ser possível criar um usuário com e-mail já utilizado;
- Caso não seja encontrado usuário com o ID informado no PUT, um novo usuário deverá ser criado;
- Não deve ser possível cadastrar usuário com e-mail já utilizado utilizando PUT;
- Os testes executados deverão conter evidências;
- Não deverá ser possível cadastrar usuários com e-mails de provedor gmail e hotmail;
- Os e-mails devem seguir um padrão válido de e-mail para o cadastro;
- As senhas devem possuír no mínimo 5 caracteres e no máximo 10 caracteres;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

US 002: [API] Login

Sendo um vendedor de uma loja com cadastro já realizado Gostaria de poder me autenticar no Marketplace da ServeRest Para poder cadastrar, editar, atualizar e excluir meus produtos

DoR (Definition of Ready)

- Banco de dados e infraestrutura para desenvolvimento disponibilizados;
- API de cadastro de usuários implementada;
- Ambiente de testes disponibilizado.

DoD (Definition of Done)

- Autenticação com geração de token Bearer implementada;
- Análise de testes cobrindo a rota de login;
- Matriz de rastreabilidade atualizada;
- Automação de testes baseado na análise realizada;

Acceptance Criteria (Critérios de Aceite)

- Usuários não cadastrados não deverão conseguir autenticar;
- Usuários com senha inválida não deverão conseguir autenticar;
- No caso de não autenticação, deverá ser retornado um status code 401 (Unauthorized);
- Usuários existentes e com a senha correta deverão ser autenticados;
- A autenticação deverá gerar um token Bearer;
- A duração da validade do token deverá ser de 10 minutos;
- Os testes executados deverão conter evidências;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

US 003: [API] Produtos

Sendo um vendedor de uma loja com cadastro já realizado Gostaria de poder me autenticar e cadastrar produtos no Marketplace do ServeRest Para poder cadastrar, editar, atualizar e excluir meus produtos

DoR (Definition of Ready)

- Banco de dados e infraestrutura para desenvolvimento disponibilizados;
- API de cadastro de usuários implementada;

- API de autenticação implementada;
- Ambiente de testes disponibilizado.

DoD (Definition of Done)

- CRUD de cadastro de Produtos implementado (CRIAR, ATUALIZAR, LISTAR E DELETAR);
- Análise de testes cobrindo a rota de produtos;
- Matriz de rastreabilidade atualizada;
- Automação de testes baseado na análise realizada;

Acceptance Criteria (Critérios de Aceite)

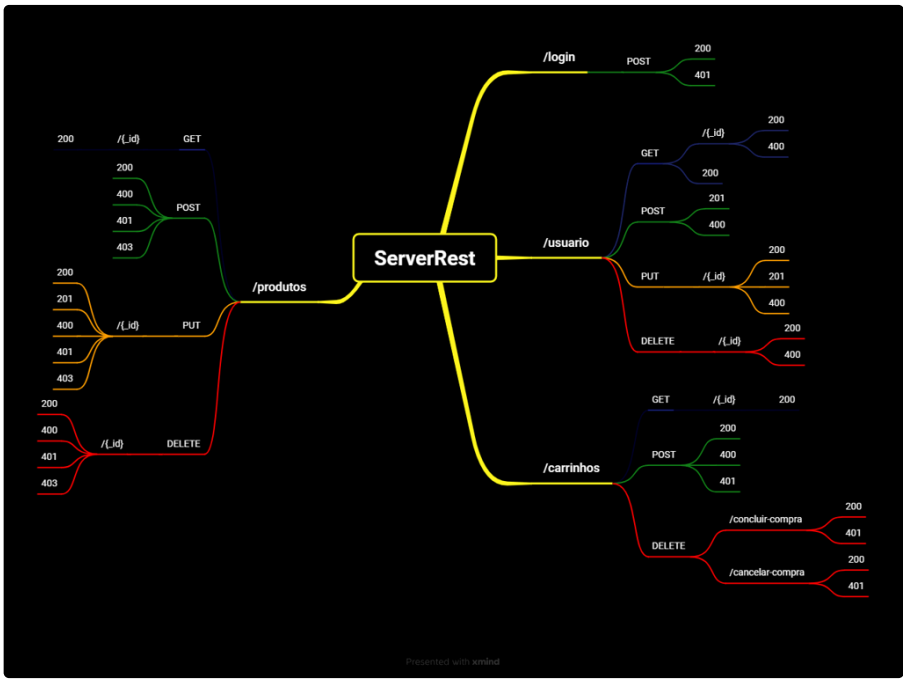
- Usuários não autenticados não devem conseguir realizar ações na rota de Produtos;
- Não deve ser possível realizar o cadastro de produtos com nomes já utilizados;
- Não deve ser possível excluir produtos que estão dentro de carrinhos (dependência API Carrinhos);
- Caso não exista produto com o o ID informado na hora do UPDATE, um novo produto deverá ser criado;
- Produtos criados através do PUT não poderão ter nomes previamente cadastrados;
- Os testes executados deverão conter evidências;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

5. Técnicas e Abordagens de Teste Aplicadas

- **Testes Funcionais e de Validação:** Esta será a base da estratégia, consistindo na verificação sistemática de cada endpoint com seus respectivos métodos HTTP para garantir que as operações CRUD funcionem conforme especificado na documentação. Serão executados testes de "caminho feliz" para validar as operações bem-sucedidas e a correção dos dados retornados.
- **Testes de Contrato (Schema Validation):** A documentação Swagger/OpenAPI da API será tratada como um contrato formal. Cada resposta da API, especialmente em cenários de sucesso, será validada programaticamente contra o schema JSON esperado. Esta técnica é crucial para garantir a consistência da API e prevenir que alterações na estrutura das respostas quebrem as integrações dos consumidores sem aviso prévio.
- **Testes de Tratamento de Erros:** Esta abordagem foca em provocar deliberadamente condições de erro para validar a robustez e a clareza da API ao lidar com falhas. Serão testadas sistematicamente as respostas para requisições com dados inválidos, parâmetros ausentes, ou violações de regras de negócio, verificando se a API retorna os códigos de status HTTP apropriados (e.g., `400 Bad Request`, `401 Unauthorized`, `404 Not Found`) e se as mensagens de erro são informativas e úteis para o desenvolvedor consumidor.

- **Testes de Segurança (Nível de API):** Os testes de segurança se concentrarão em dois pilares fundamentais da API:
  - **Autenticação:** Serão enviados requests para endpoints protegidos sem o token de autorização, com um token inválido (malformado ou expirado) ou com um token de um usuário inexistente. O resultado esperado é sempre uma resposta de não autorizado (`401 Unauthorized`).
  - **Autorização:** Serão realizadas tentativas de executar ações restritas a administradores (como cadastrar um novo produto) utilizando o token de um usuário comum. O resultado esperado é uma resposta de acesso proibido (`403 Forbidden`).
- **Testes de Limite:** Focará em validar o comportamento da API ao receber entradas nos limites das regras de negócio definidas. Isso inclui, por exemplo, testar os valores mínimos e máximos permitidos para campos numéricos e de texto (como senhas), e verificar como o sistema reage a valores que estão exatamente no limite ou logo acima/abaixo dele.

6. Mapa mental da aplicação



7. Cenários de Teste Detalhados

Os cenários de teste serão documentados de forma estruturada para garantir clareza, rastreabilidade e facilidade de execução, tanto manual quanto automatizada.

Endpoint: /login

- **ID do Cenário:** LOG-001

- **Descrição:** Realizar login com sucesso (Administrador).
- **Pré-condições:** Um usuário administrador existe no banco de dados.
- **Dados de Entrada (Payload JSON):** `{ "email": "admin@email.com", "password": "senha_valida" }`
- **Resultado Esperado: Status:** `200 OK`. **Corpo:** Retorna a mensagem "Login realizado com sucesso" e um token de autorização.
- **Prioridade:** Alta

- **ID do Cenário:** LOG-002
  - **Descrição:** Realizar login com sucesso (Usuário Comum).
  - **Pré-condições:** Um usuário comum existe no banco de dados.
  - **Dados de Entrada (Payload JSON):** `{ "email": "comum@email.com", "password": "senha_valida" }`
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Retorna a mensagem "Login realizado com sucesso" e um token de autorização.
  - **Prioridade:** Alta
- **ID do Cenário:** LOG-003
  - **Descrição:** Tentar login com senha incorreta.
  - **Pré-condições:** Um usuário existe com uma senha diferente da informada.
  - **Dados de Entrada (Payload JSON):** `{ "email": "usuario@email.com", "password": "senha_incorreta" }`
  - **Resultado Esperado: Status:** `401 Unauthorized`. **Corpo:** Mensagem "Email e/ou senha inválidos".
  - **Prioridade:** Alta
- **ID do Cenário:** LOG-004
  - **Descrição:** Tentar login com e-mail não cadastrado.
  - **Pré-condições:** O e-mail informado não existe no banco de dados.
  - **Dados de Entrada (Payload JSON):** `{ "email": "inexistente@email.com", "password": "qualquer_senha" }`
  - **Resultado Esperado: Status:** `401 Unauthorized`. **Corpo:** Mensagem "Email e/ou senha inválidos".
  - **Prioridade:** Alta

- **ID do Cenário:** LOG-005
  - **Descrição:** Tentar login com campo `email` vazio.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada (Payload JSON):** `{ "email": "", "password": "senha_valida" }`
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "email não pode ficar em branco".
  - **Prioridade:** Média
- **ID do Cenário:** LOG-006
  - **Descrição:** Tentar login com campo `password` ausente.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada (Payload JSON):** `{ "email": "usuario@email.com" }`
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "password é obrigatório".
  - **Prioridade:** Média
- **ID do Cenário:** LOG-007
  - **Descrição:** Tentar login com formato de e-mail inválido.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada (Payload JSON):** `{ "email": "email-invalido", "password": "senha_valida" }`
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "email deve ser um email válido".
  - **Prioridade:** Média
- **ID do Cenário:** LOG-008
  - **Descrição:** Tentar login enviando um payload com campos extras.
  - **Pré-condições:** Um usuário válido existe.
  - **Dados de Entrada (Payload JSON):** `{ "email": "usuario@email.com", "password": "senha_valida", "campo_extra": "valor" }`
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** A API deve ignorar o campo extra e retornar o token normalmente.
  - **Prioridade:** Baixa

Endpoint: /usuarios

- **ID do Cenário:** USR-001
  - **Descrição:** Cadastrar novo usuário comum com sucesso.
  - **Pré-condições:** E-mail do novo usuário não existe no banco.
  - **Dados de Entrada: Payload:** `{ "nome": "Usuario Comum", "email": "novo_comum@email.com", "password": "senha123", "administrador": "false" }`
  - **Resultado Esperado: Status:** `201 Created`. **Corpo:** Mensagem "Cadastro realizado com sucesso" e o `_id` do novo usuário.
  - **Prioridade:** Alta
- **ID do Cenário:** USR-002
  - **Descrição:** Listar todos os usuários cadastrados.
  - **Pré-condições:** Existem usuários no banco de dados.
  - **Dados de Entrada:** N/A (Requisição GET para `/usuarios`).
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Retorna uma lista contendo todos os usuários.
  - **Prioridade:** Alta
- **ID do Cenário:** USR-003
  - **Descrição:** Tentar cadastrar usuário com e-mail já existente.
  - **Pré-condições:** Um usuário com o e-mail "existente@email.com" já foi cadastrado.
  - **Dados de Entrada: Payload:** `{ "nome": "Outro Nome", "email": "existente@email.com", "password": "senha123", "administrador": "false" }`
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Este email já está sendo usado".
  - **Prioridade:** Alta
- **ID do Cenário:** USR-004
  - **Descrição:** Excluir um usuário com sucesso.
  - **Pré-condições:** Um usuário com o ID a ser excluído existe. Autenticado como admin.
  - **Dados de Entrada:** N/A (Requisição DELETE para `/usuarios/{id}`).
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Mensagem "Registro excluído com sucesso".
  - **Prioridade:** Alta

- **ID do Cenário:** USR-005
  - **Descrição:** Tentar cadastrar usuário com senha abaixo do limite.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada: Payload:** `{ ... "password": "1234" }` (4 caracteres).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem sobre o tamanho mínimo da senha.
  - **Prioridade:** Média
- **ID do Cenário:** USR-006
  - **Descrição:** Tentar cadastrar usuário com senha acima do limite.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada: Payload:** `{ ... "password": "12345678901" }` (11 caracteres).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem sobre o tamanho máximo da senha.
  - **Prioridade:** Média
- **ID do Cenário:** USR-007
  - **Descrição:** Atualizar dados de um usuário existente (PUT).
  - **Pré-condições:** Um usuário com o ID a ser atualizado existe. Autenticado como admin.
  - **Dados de Entrada: Payload:** `{ "nome": "Nome Atualizado", ... }`
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Mensagem "Registro alterado com sucesso".
  - **Prioridade:** Média
- **ID do Cenário:** USR-008
  - **Descrição:** Buscar um usuário por ID inexistente.
  - **Pré-condições:** O ID informado não corresponde a nenhum usuário no banco.
  - **Dados de Entrada:** N/A (Requisição GET para `/usuarios/id_inexistente`).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Usuário não encontrado".
  - **Prioridade:** Média

Endpoint: /produtos

- **ID do Cenário:** PROD-001
  - **Descrição:** Cadastrar novo produto com sucesso.
  - **Pré-condições:** Usuário autenticado como `administrador`.

    - **Dados de Entrada: Payload:** `{ "nome": "Produto Novo", "preco": 100, "descricao": "Desc", "quantidade": 50 }`
    - **Resultado Esperado: Status:** `201 Created`. **Corpo:** Mensagem "Cadastro realizado com sucesso" e o `_id` do novo produto.
    - **Prioridade:** Alta
- **ID do Cenário:** PROD-002
    - **Descrição:** Tentar cadastrar produto sem autenticação.
    - **Pré-condições:** Nenhuma.
    - **Dados de Entrada:** Payload JSON válido.
    - **Resultado Esperado: Status:** `401 Unauthorized`. **Corpo:** Mensagem "Token de acesso ausente, inválido ou expirado".
    - **Prioridade:** Alta
- **ID do Cenário:** PROD-003
    - **Descrição:** Tentar cadastrar produto com usuário comum.
    - **Pré-condições:** Usuário autenticado como `comum` (não-admin).
    - **Dados de Entrada:** Payload JSON válido.
    - **Resultado Esperado: Status:** `403 Forbidden`. **Corpo:** Mensagem "Rota exclusiva para administradores".
    - **Prioridade:** Alta
- **ID do Cenário:** PROD-004
    - **Descrição:** Tentar cadastrar produto com nome duplicado.
    - **Pré-condições:** Um produto com o mesmo nome já existe. Usuário admin autenticado.
    - **Dados de Entrada:** Payload JSON com nome de produto já existente.
    - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Já existe produto com esse nome".
    - **Prioridade:** Alta
- **ID do Cenário:** PROD-005
    - **Descrição:** Listar todos os produtos cadastrados.
    - **Pré-condições:** Existem produtos no banco de dados.
    - **Dados de Entrada:** N/A (Requisição GET para `/produtos`).
    - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Retorna uma lista contendo todos os produtos.
    - **Prioridade:** Alta

- **ID do Cenário:** PROD-006
  - **Descrição:** Tentar cadastrar produto com campo obrigatório ausente.
  - **Pré-condições:** Usuário admin autenticado.
  - **Dados de Entrada: Payload:** `{ "preco": 100, "descricao": "Desc", "quantidade": 50 }` (sem o campo `nome`).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "nome é obrigatório".
  - **Prioridade:** Média
- **ID do Cenário:** PROD-007
  - **Descrição:** Tentar cadastrar produto com tipo de dado inválido.
  - **Pré-condições:** Usuário admin autenticado.
  - **Dados de Entrada: Payload:** `{ "nome": "Produto Inválido", "preco": "cem_reais", ... }` (`preco` como string).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "preco deve ser um número".
  - **Prioridade:** Média
- **ID do Cenário:** PROD-008
  - **Descrição:** Tentar excluir um produto que faz parte de um carrinho.
  - **Pré-condições:** O produto a ser excluído está em um carrinho ativo. Usuário admin autenticado.
  - **Dados de Entrada:** N/A (Requisição DELETE para `/produtos/{id}`).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Não é permitido excluir produto que faz parte de carrinho".
  - **Prioridade:** Alta

Endpoint: /carrinhos

- **ID do Cenário:** CAR-001
  - **Descrição:** Criar um carrinho com um produto com sucesso.
  - **Pré-condições:** Usuário comum autenticado. Produto existe em estoque.
  - **Dados de Entrada: Payload:** `{ "produtos": [{ "idProduto": "id_valido", "quantidade": 1 }] }`
  - **Resultado Esperado: Status:** `201 Created`. **Corpo:** Mensagem "Cadastro realizado com sucesso" e `_id` do carrinho.
  - **Prioridade:** Alta

- **ID do Cenário:** CAR-002
  - **Descrição:** Concluir uma compra com sucesso e verificar baixa no estoque.
  - **Pré-condições:** Usuário autenticado possui um carrinho com produtos. Estoque é suficiente.
  - **Dados de Entrada:** N/A (Requisição DELETE para `/carrinhos/concluir-compra`).
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Mensagem "Registro excluído com sucesso". **Validação extra:** Estoque do produto em `/produtos` foi decrementado.
  - **Prioridade:** Alta
- **ID do Cenário:** CAR-003
  - **Descrição:** Tentar concluir compra com estoque insuficiente.
  - **Pré-condições:** Carrinho possui um item com quantidade maior que o estoque disponível.
  - **Dados de Entrada:** N/A (Requisição DELETE para `/carrinhos/concluir-compra`).
  - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Produto com estoque insuficiente". **Validação extra:** Estoque do produto não foi alterado.
  - **Prioridade:** Alta
- **ID do Cenário:** CAR-004
  - **Descrição:** Cancelar uma compra e verificar que o estoque não foi alterado.
  - **Pré-condições:** Usuário autenticado possui um carrinho com produtos.
  - **Dados de Entrada:** N/A (Requisição DELETE para `/carrinhos/cancelar-compra`).
  - **Resultado Esperado: Status:** `200 OK`. **Corpo:** Mensagem "Registro excluído com sucesso". **Validação extra:** Estoque dos produtos permanece inalterado.
  - **Prioridade:** Alta
- **ID do Cenário:** CAR-005
  - **Descrição:** Tentar criar um carrinho sem estar autenticado.
  - **Pré-condições:** Nenhuma.
  - **Dados de Entrada:** Payload JSON válido.
  - **Resultado Esperado: Status:** `401 Unauthorized`. **Corpo:** Mensagem "Token de acesso ausente, inválido ou expirado".
  - **Prioridade:** Alta
- **ID do Cenário:** CAR-006
  - **Descrição:** Tentar adicionar um produto com ID inexistente ao carrinho.
  - **Pré-condições:** Usuário autenticado.

    - **Dados de Entrada: Payload:** `{ "produtos": [{ "idProduto": "id_inexistente", "quantidade": 1 }] }`
    - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Produto não encontrado".
    - **Prioridade:** Média
- **ID do Cenário:** CAR-007
    - **Descrição:** Tentar adicionar um produto com quantidade zero ou negativa.
    - **Pré-condições:** Usuário autenticado. Produto existe.
    - **Dados de Entrada: Payload:** `{ "produtos": [{ "idProduto": "id_valido", "quantidade": 0 }] }`
    - **Resultado Esperado: Status:** `400 Bad Request`. **Corpo:** Mensagem "Não é permitido adicionar produto com quantidade zero ou menor".
    - **Prioridade:** Média
- **ID do Cenário:** CAR-008
    - **Descrição:** Tentar excluir (concluir/cancelar) um carrinho inexistente.
    - **Pré-condições:** Usuário autenticado, mas não possui carrinho ativo.
    - **Dados de Entrada:** N/A (Requisição DELETE para `/carrinhos/concluir-compra`).
    - **Resultado Esperado: Status:** `404 Not Found`. **Corpo:** Mensagem "Carrinho não encontrado".
    - **Prioridade:** Média

8. Priorização da Execução dos Testes

Os cenários serão classificados em três níveis de prioridade:

- **Prioridade Alta:** Este nível inclui todos os testes de "caminho feliz" para os fluxos de negócio essenciais (login, cadastro de usuário, busca de produtos e o fluxo completo de compra). Também abrange todos os testes de segurança fundamentais, como validação de autenticação e autorização.
- **Prioridade Média:** Este nível engloba testes para cenários alternativos e o tratamento de erros comuns e esperados, como tentativas de cadastrar dados duplicados ou o envio de dados inválidos em campos críticos.
- **Prioridade Baixa:** Este nível inclui validações de campos menos críticos, que têm menor probabilidade de ocorrer ou menor impacto no negócio.

ID do Risco	Descrição do Risco	Categoria	Probabilidade (1-5)	Impacto (1-5)	Nível de Risco (P*I)	Mitigação (Cenários de Teste)
R-01	Acesso não autorizado a funcionalidades de administrador devido a falha na validação do perfil do usuário.	Segurança	2 (Baixa)	5 (Crítico)	10 (Alto)	PROD-003, e cenários equivalentes para outros endpoints de admin.
R-02	Cadastro de dados inválidos (e.g., e-mail de usuário duplicado) corrompe o estado do sistema.	Integridade de Dados	3 (Média)	4 (Alto)	12 (Alto)	PROD-004.

R-03	Exclusão de um usuário não remove seus carrinhos de compras associados, deixando dados órfãos no banco de dados.	Integridade de Dados	2 (Baixa)	3 (Média)	6 (Médio)	Cenários de exclusão de usuário seguidos pela verificação de seus dados relacionados.

10. Estratégia de Cobertura de Testes

O foco será em métricas de "caixa-preta" específicas para APIs REST, que medem a abrangência dos testes em relação ao comportamento externo e ao contrato da aplicação.

Métricas

Path Coverage (Cobertura de Endpoints): Mede a porcentagem de endpoints e métodos HTTP documentados que foram cobertos pelos testes.

Status Code Coverage (Cobertura de Códigos de Status): Mede a variedade de códigos de status HTTP que foram validados para cada endpoint, cobrindo tanto cenários de sucesso quanto de falha.

Parameter Coverage (Cobertura de Parâmetros): Mede a cobertura dos parâmetros de entrada (query, path, e campos do corpo da requisição).

11. Análise de Candidatos à Automação

Critérios de Seleção para Automação

- **Repetitividade:** Testes que precisam ser executados com frequência, como testes de regressão a cada nova build.

- **Criticidade de Negócio:** Testes que cobrem os fluxos de negócio mais críticos, cuja falha teria um impacto significativo.
- **Propensão a Regressão:** Funcionalidades complexas ou que são interdependentes, onde uma alteração pode facilmente introduzir defeitos em outras partes do sistema.
- **Determinismo:** Testes que produzem resultados consistentes e previsíveis, sem depender de fatores externos variáveis.

Conjuntos de Testes Candidatos à Automação

Com base nos critérios acima, os seguintes conjuntos de testes são os principais candidatos à automação:

- **Suíte de Smoke Test:** Composta por todos os cenários de Prioridade Alta (P1). Esta suíte deve ser leve e de execução rápida, ideal para ser integrada ao pipeline de CI/CD e executada a cada commit para fornecer feedback imediato aos desenvolvedores.
- **Suíte de Regressão Funcional:** Uma suíte mais completa, incluindo todos os cenários de Prioridade Alta e Média. Esta suíte garante uma cobertura mais ampla e pode ser executada em um ambiente de staging, em uma cadência regular (e.g., diariamente) ou antes de cada implantação em produção.

12. Ferramentas e Ambiente de Teste

A automação dos cenários de teste descritos neste plano será desenvolvida utilizando o Robot Framework, um framework de automação de testes open-source. O ecossistema de ferramentas inclui:

12.1. Automação de Testes

- **Framework Principal:** Robot Framework.
- **Biblioteca de API:** `RequestsLibrary`, para realizar as chamadas HTTP (GET, POST, PUT, DELETE) e validar as respostas.
- **Bibliotecas de Suporte:** `Collections` (para manipulação de dicionários JSON) e `OperatingSystem` (para gestão de variáveis de ambiente).
- **Ambiente de Execução:** Python 3.11.

12.2. Gestão de Testes e Defeitos

- **Plataforma de Gestão:** Jira Software. Será utilizado para o gerenciamento de User Stories, tarefas e o ciclo de vida dos defeitos (bugs).
- **Plugin de Gestão de Testes:** QAlity for Jira. Será utilizado para a criação, organização e execução dos casos de teste diretamente no Jira, garantindo a rastreabilidade com os requisitos.

13. Estrutura do Projeto de Automação

O projeto de automação seguirá uma estrutura organizada para promover a reutilização de código e a manutenibilidade:

- **/tests:** Contém os arquivos `.robot` com os casos de teste, separados por suíte (ex., `usuarios.robot`, `produtos.robot`).
- **/resources:** Contém arquivos `.robot` com keywords reutilizáveis (ex, `auth_keywords.robot` para encapsular a lógica de login e obtenção de token) e arquivos de variáveis.
- **/results:** Pasta padrão onde o Robot Framework salvará os relatórios de execução.

14. Gestão de Evidências e Relatório de Defeitos

14.1. Coleta de Evidências: As evidências da execução dos testes automatizados serão os **relatórios** `report.html` **e** `log.html` gerados nativamente pelo Robot Framework.

- O `report.html` fornecerá uma visão geral e estatística dos resultados da execução.
- O `log.html` servirá como evidência detalhada, mostrando cada passo executado, as requisições, as respostas e o resultado de cada validação, sendo fundamental para a análise de falhas.

14.2. Relatório de Defeitos:

- Todo defeito identificado será registrado no Jira com o auxilio do puglin QALity.
- O relatório do defeito conterá:
  - Um título claro e conciso.
  - Passos para reprodução.
  - O resultado esperado vs. o resultado obtido.
  - A evidência anexada (log ou captura de tela).
  - A prioridade do defeito (baseada na matriz de risco da Seção 9).

15. Gestão do Ciclo de Vida dos Testes (Jira + QAlity)

O processo de teste seguirá um ciclo de vida estruturado dentro do Jira para garantir organização e rastreabilidade ponta a ponta.

1. **Criação dos Casos de Teste:** Todos os cenários detalhados neste plano serão cadastrados como itens do tipo "Test Case" no QAlity. Cada caso de teste será diretamente vinculado à sua respectiva User Story no Jira.

2. **Organização em Ciclos de Teste:** Os casos de teste serão agrupados em "Test Cycles" no QAlity para organizar as execuções. Exemplos de ciclos incluem:

- **Smoke Tests:** Executado a cada nova build no ambiente de desenvolvimento.
- **Regressão Funcional:** Executado antes de uma implantação para o ambiente de staging/produção.
- **Testes de Borda:** Executado sob demanda para validar funcionalidades específicas.

3. **Execução e Registro de Resultados:** A execução dos testes automatizados com Robot Framework será refletida no QAlity. O status de cada caso de teste (`PASS` , `FAIL` , `BLOCKED`) será atualizado no ciclo de teste correspondente. As evidências (relatórios `log.html`) serão anexadas ao registro da execução.