Planejamento de Testes - API Serverest

Plano de Testes: API REST ServeRest

Data de Elaboração: 05 de Setembro de 2025

1. Apresentação

Este documento detalha o plano de testes para a API REST da aplicação ServeRest, acessível através do endpoint base https://compassuol.serverest.dev/. O foco deste planejamento é garantir a qualidade, a conformidade com as regras de negócio e a estabilidade de todos os endpoints da API, incluindo **Usuários, Login, Produtos e Carrinhos**, utilizando como base a documentação Swagger, as User Stories e o mapa mental da aplicação.

Este plano servirá como guia para a execução dos testes, o mapeamento de issues e a criação de uma collection Postman para validação contínua.

2. Objetivo

O objetivo principal é validar de forma sistemática todos os endpoints da API ServeRest para garantir que eles operem conforme o esperado.

Objetivos Específicos:

- Validar o CRUD completo do endpoint `/usuarios`.
- Verificar o processo de autenticação no endpoint `/login`.
- Validar o CRUD completo do endpoint `/produtos`, incluindo a necessidade de autenticação.
- Validar o fluxo completo do endpoint `/carrinhos`, desde a criação até a conclusão ou cancelamento da compra.
- Garantir que todas as regras de negócio e critérios de aceitação sejam atendidos.
- Identificar, documentar e reportar quaisquer divergências (issues) entre o comportamento esperado e o real.
- Produzir uma collection Postman com scripts de validação para automação dos testes.

3. Escopo

Endpoints e Funcionalidades em Escopo:

- **Usuários (`/usuarios`):** CRUD completo (POST, GET, PUT, DELETE).
- **Login (`/login`):** Autenticação (POST).
- **Produtos (`/produtos`):** CRUD completo (POST, GET, PUT, DELETE).
- **Carrinhos (`/carrinhos`):**
  - `GET /carrinhos` e `GET /carrinhos/{_id}`: Listagem e busca de carrinhos.
  - `POST /carrinhos`: Criação de um novo carrinho (requer autenticação).
  - `DELETE /carrinhos/concluir-compra`: Finalização de uma compra (requer autenticação).
  - `DELETE /carrinhos/cancelar-compra`: Cancelamento de uma compra e retorno dos produtos ao estoque (requer autenticação).

Fora de Escopo:

- A aplicação front-end (Front - ServeRest).

- Testes de performance, carga ou estresse.
- Testes de infraestrutura ou segurança aprofundados.

4. Análise

A análise será baseada na comparação direta entre os resultados obtidos nas chamadas da API e o comportamento esperado, definido pelas User Stories e pelo Swagger. Serão avaliados:

- **Status Codes HTTP:** Conformidade com os padrões REST.
- **Corpo da Resposta (Response Body):** Validação da estrutura do JSON e dos dados.
- **Headers da Resposta:** Verificação de headers importantes como `Authorization`.
- **Regras de Negócio:** Validação de lógicas específicas (unicidade de e-mail, controle de estoque, etc.).

5. Técnicas Aplicadas

- **Teste Baseado em Especificação:** Uso das User Stories e Swagger.
- **Análise de Valor Limite:** Para campos com restrições (ex: tamanho da senha).
- **Particionamento de Equivalência:** Para campos com regras (ex: formato e domínio do e-mail).
- **Teste de Transição de Estado:** Para validar fluxos sequenciais (Login → Criar Carrinho → Concluir Compra).
- **Teste de Robustez (Fuzzing):** Envio de dados malformados, inesperados ou de tipos incorretos para validar o tratamento de erros.

6. Ambiente e Recursos de Teste (API)

Local dos Testes

- Local de trabalho pessoal.
- Ambiente utilizado na plataforma Air Learning, podendo ser realizado remotamente.

Ambiente de Testes (Hardware e Software)

- **Sistema Operacional:** Windows 11 Home Single Language (Versão 24H2)
- **Hardware:** PC (Intel i5, 12GB RAM, 64 bits)
- **Software:**
  - Ferramenta de Teste de API: Postman
  - Navegador (para consulta): Google Chrome

Recursos Necessários

- **Humanos:** Testadores.
- **Equipamentos:** Computadores com acesso à internet.
- **Recursos Financeiros:** Não consta.

7. Mapa Mental da Aplicação (Estrutura da API)

```
API ServeRest
├── Endpoint: /login
│    └── POST
│         ├── Sucesso -> 200 OK
│         └── Falha   -> 401 Unauthorized
│
├── Endpoint: /usuarios
│    ├── POST (Criar)
│    │    ├── Sucesso -> 201 Created
│    │    └── Falha   -> 400 Bad Request (E-mail duplicado, dados inválidos)
│    ├── GET (Listar / Buscar por ID)
```

```
|     |     └── Sucesso -> 200 OK
|     ├── PUT /{_id} (Atualizar/Criar)
|     |     ├── Atualizado -> 200 OK
|     |     └── Criado     -> 201 Created
|     └── DELETE /{_id} (Excluir)
|           ├── Sucesso -> 200 OK
|           └── Falha   -> 400 Bad Request (Usuário com carrinho)
|
├── Endpoint: /produtos
|     ├── POST (Criar)
|     |     ├── Sucesso      -> 201 Created
|     |     ├── Falha (Nome) -> 400 Bad Request
|     |     ├── Falha (Auth) -> 401 Unauthorized
|     |     └── Falha (Perm) -> 403 Forbidden
|     ├── GET (Listar / Buscar por ID)
|     |     └── Sucesso -> 200 OK
|     ├── PUT /{_id} (Atualizar/Criar)
|     |     ├── Atualizado   -> 200 OK
|     |     └── Criado       -> 201 Created
|     └── DELETE /{_id} (Excluir)
|           ├── Sucesso -> 200 OK
|           └── Falha   -> 400 Bad Request (Produto em carrinho)
|
└── Endpoint: /carrinhos
      ├── POST (Criar)
      |     ├── Sucesso -> 201 Created
      |     ├── Falha   -> 400 Bad Request (Produto s/ estoque, etc.)
      |     └── Falha   -> 401 Unauthorized
      ├── GET (Listar / Buscar por ID)
      |     └── Sucesso -> 200 OK
      ├── DELETE /concluir-compra
      |     ├── Sucesso -> 200 OK
      |     └── Falha   -> 401 Unauthorized
      └── DELETE /cancelar-compra
            ├── Sucesso -> 200 OK
            └── Falha   -> 401 Unauthorized
```

8. Cenários de Teste Detalhados (BDD/Gherkin)

Esta seção contém cenários para todos os métodos da API.

US 001: [API] Usuários

Feature: Gerenciamento de Usuários

Cenários POST

Cenário: Criar um novo usuário com sucesso

```
Dado que eu possuo os dados de um novo usuário administrador
Quando eu envio uma requisição POST para a rota /usuarios com esses dados
Então a resposta deve ter o status code 201
E a resposta deve conter a mensagem "Cadastro realizado com sucesso"


```

Cenário: Tentar criar um usuário com um e-mail duplicado

```
Dado que já existe um usuário cadastrado com o e-mail "email.duplicado@exemplo.com"
Quando eu envio uma requisição POST para a rota /usuarios com o mesmo e-mail
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Este email já está sendo usado"


```

Sub-Feature: Testes de Robustez e Dados Inválidos (POST /usuarios)

Cenário: Tentar criar um usuário sem um campo obrigatório

```
Dado que eu vou criar um novo usuário
Quando eu envio uma requisição POST para a rota /usuarios sem o campo "email"
Então a resposta deve ter o status code 400
E o corpo da resposta deve indicar que o campo "email" é obrigatório


```

Cenário: Testar a criação de usuário com tipos de dados incorretos no payload

```
Dado que eu vou criar um novo usuário
Quando eu envio uma requisição POST para a rota /usuarios com o campo "nome" sendo o número 123 em vez de uma string
Então a resposta deve ter o status code 400
E o corpo da resposta deve indicar que o campo "nome" deve ser do tipo string


```

Cenário de Esquema: Testar a criação de usuário com dados de string malformados ou inesperados

```
Dado que eu vou criar um novo usuário
Quando eu envio uma requisição POST para a rota /usuarios com o campo "<campo>" preenchido com "<valor_invalido>"
Então a resposta deve ter o status code 400
E o corpo da resposta deve indicar que o campo "<campo>" é inválido

Exemplos:
| campo    | valor_invalido                                          | descrição                          |
|----------|---------------------------------------------------------|------------------------------------|
| nome     | (uma string com mais de 255 caracteres)                 | String Longa                       |
| nome     | ""                                                      | String Vazia (em branco)           |
| nome     | "´select * from customer"                               | Injeção de SQL                     |
| password | "1234"                                                  | Senha muito curta                  |
```

```
| password | "12345678901"                                    |
Senha muito longa                    |
| email    | "email-invalido"                                 | E-
mail sem @                          |
| email    | "usuario@gmail.com"                              | E-
mail com domínio proibido           |


```

Cenários GET

Cenário: Listar todos os usuários cadastrados

```
Dado que existem múltiplos usuários cadastrados no sistema
Quando eu envio uma requisição GET para a rota /usuarios
Então a resposta deve ter o status code 200
E o corpo da resposta deve ser uma lista contendo todos os usuários


```

Cenário: Buscar um usuário por um ID existente

```
Dado que existe um usuário cadastrado com um ID conhecido
Quando eu envio uma requisição GET para a rota /usuarios/{id} com o ID
conhecido
Então a resposta deve ter o status code 200
E o corpo da resposta deve conter os dados do usuário específico


```

Cenário: Tentar buscar um usuário por um ID inexistente

```
Dado que um ID de usuário não existe no sistema
Quando eu envio uma requisição GET para a rota /usuarios/{id} com o ID
inexistente
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Usuário não encontrado"


```

Cenários PUT

Cenário: Atualizar um usuário existente com sucesso

```
Dado que existe um usuário cadastrado com um ID conhecido
Quando eu envio uma requisição PUT para a rota /usuarios/{id} com novos
dados válidos
Então a resposta deve ter o status code 200
E a resposta deve conter a mensagem "Registro alterado com sucesso"


```

Cenário: Criar um novo usuário ao tentar atualizar um ID inexistente

```
Dado que um ID de usuário não existe no sistema
Quando eu envio uma requisição PUT para a rota /usuarios/{id} com dados
de um novo usuário
Então a resposta deve ter o status code 201
E a resposta deve conter a mensagem "Cadastro realizado com sucesso"


```

Cenários DELETE

Cenário: Excluir um usuário existente com sucesso

```
Dado que existe um usuário cadastrado sem carrinho
Quando eu envio uma requisição DELETE para a rota /usuarios/{id} deste
usuário
Então a resposta deve ter o status code 200
```

```
E a resposta deve conter a mensagem "Registro excluído com sucesso"


```

Cenário: Tentar excluir um usuário que possui um carrinho

```
Dado que um usuário com carrinho ativo existe
Quando eu envio uma requisição DELETE para a rota /usuarios/{id} desse usuário
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Não é permitido excluir usuário com carrinho cadastrado"


```

US 002: [API] Login

Feature: Autenticação de Usuário

Cenário: Realizar login com sucesso

```
Dado que existe um usuário cadastrado com e-mail e senha válidos
Quando eu envio uma requisição POST para a rota /login com essas credenciais
Então a resposta deve ter o status code 200
E o corpo da resposta deve conter um campo authorization


```

Cenário: Tentar realizar login com credenciais inválidas

```
Dado que um usuário tenta se autenticar com e-mail não cadastrado
Quando eu envio uma requisição POST para a rota /login
Então a resposta deve ter o status code 401
E a resposta deve conter a mensagem "Email e/ou senha inválidos"


```

Cenário: Validar expiração do token de autenticação

```
Dado que eu realizei o login e obtive um token de autenticação
Quando eu aguardo o tempo de expiração do token (600 segundos)
E tento acessar uma rota protegida, como POST /produtos
Então a resposta deve ter o status code 401
E a resposta deve conter uma mensagem sobre token expirado


```

US 003: [API] Produtos

Feature: Gerenciamento de Produtos

Cenários POST

Cenário: Cadastrar um produto com sucesso

```
Dado que eu estou autenticado como um administrador
Quando eu envio uma requisição POST para a rota /produtos com dados de um novo produto
Então a resposta deve ter o status code 201
E a resposta deve conter a mensagem "Cadastro realizado com sucesso"


```

Cenário: Tentar cadastrar um produto sem autenticação

```
Dado que eu não estou autenticado
Quando eu envio uma requisição POST para a rota /produtos
Então a resposta deve ter o status code 401
E a resposta deve conter uma mensagem sobre token ausente ou inválido
```

```
5
6
```

Cenário: Tentar cadastrar um produto com um nome já existente

```
Dado que eu estou autenticado como um administrador
E já existe um produto com o nome "Produto Duplicado"
Quando eu envio uma requisição POST para a rota /produtos com o nome
"Produto Duplicado"
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Já existe produto com esse nome"


```

Cenário: Tentar cadastrar um produto com um usuário não-administrador

```
Dado que eu estou autenticado como um usuário comum (não administrador)
Quando eu envio uma requisição POST para a rota /produtos
Então a resposta deve ter o status code 403
E a resposta deve conter a mensagem "Rota exclusiva para
administradores"


```

Cenários GET

Cenário: Listar todos os produtos cadastrados

```
Dado que existem múltiplos produtos cadastrados no sistema
Quando eu envio uma requisição GET para a rota /produtos
Então a resposta deve ter o status code 200
E o corpo da resposta deve ser uma lista contendo todos os produtos


```

Cenário: Buscar um produto por um ID existente

```
Dado que existe um produto cadastrado com um ID conhecido
Quando eu envio uma requisição GET para a rota /produtos/{id} com o ID
conhecido
Então a resposta deve ter o status code 200
E o corpo da resposta deve conter os dados do produto específico


```

Cenários PUT

Cenário: Atualizar um produto existente com sucesso

```
Dado que eu estou autenticado como administrador
E existe um produto cadastrado com um ID conhecido
Quando eu envio uma requisição PUT para a rota /produtos/{id} com novos
dados válidos
Então a resposta deve ter o status code 200
E a resposta deve conter a mensagem "Registro alterado com sucesso"


```

Cenário: Tentar atualizar um produto sem autenticação

```
Dado que existe um produto cadastrado com um ID conhecido
Quando eu envio uma requisição PUT para a rota /produtos/{id} sem um
token de autenticação
Então a resposta deve ter o status code 401
E a resposta deve conter a mensagem "Token de acesso ausente, inválido,
expirado ou usuário do token não existe mais"


```

Cenários DELETE

Cenário: Excluir um produto com sucesso

```
Dado que eu estou autenticado como administrador
E existe um produto que não está em nenhum carrinho
Quando eu envio uma requisição DELETE para a rota /produtos/{id} deste produto
Então a resposta deve ter o status code 200
E a resposta deve conter a mensagem "Registro excluído com sucesso"


```

Cenário: Tentar excluir um produto que está em um carrinho

```
Dado que eu estou autenticado como administrador
E existe um produto que está associado a um carrinho
Quando eu envio uma requisição DELETE para a rota /produtos/{id} deste produto
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Não é permitido excluir produto que faz parte de carrinho"


```

US 004: [API] Carrinhos

Feature: Gerenciamento de Carrinhos

Cenários POST

Cenário: Criar um carrinho de compras com sucesso

```
Dado que eu estou autenticado como um usuário comum
E existe um produto com quantidade em estoque suficiente
Quando eu envio uma requisição POST para a rota /carrinhos com o ID do produto
Então a resposta deve ter o status code 201
E a resposta deve conter a mensagem "Cadastro realizado com sucesso"


```

Cenário: Tentar criar um carrinho com produto inexistente

```
Dado que eu estou autenticado como um usuário comum
Quando eu envio uma requisição POST para a rota /carrinhos com um ID de produto que não existe
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Produto não encontrado"


```

Cenário: Tentar criar um carrinho com produto sem estoque

```
Dado que eu estou autenticado como um usuário comum
E existe um produto cuja quantidade em estoque é 0
Quando eu envio uma requisição POST para a rota /carrinhos com o ID desse produto
Então a resposta deve ter o status code 400
E a resposta deve conter a mensagem "Produto não possui quantidade suficiente"


```

Cenários GET

Cenário: Listar todos os carrinhos cadastrados

```
Dado que existem múltiplos carrinhos cadastrados no sistema
Quando eu envio uma requisição GET para a rota /carrinhos
Então a resposta deve ter o status code 200
```

```
E o corpo da resposta deve ser uma lista contendo todos os carrinhos


```

Cenário: Buscar um carrinho por um ID existente

```
Dado que existe um carrinho cadastrado com um ID conhecido
Quando eu envio uma requisição GET para a rota /carrinhos/{id} com o ID conhecido
Então a resposta deve ter o status code 200
E o corpo da resposta deve conter os dados do carrinho específico


```

Cenários DELETE

Cenário: Concluir uma compra com sucesso

```
Dado que eu estou autenticado como um usuário e possuo um carrinho ativo
Quando eu envio uma requisição DELETE para a rota /carrinhos/concluir-compra
Então a resposta deve ter o status code 200
E a resposta deve conter a mensagem "Registro excluído com sucesso"


```

Cenário: Cancelar uma compra e verificar o retorno do estoque

```
Dado que eu estou autenticado como um usuário e possuo um carrinho com um produto
E a quantidade inicial em estoque do produto é X
Quando eu envio uma requisição DELETE para a rota /carrinhos/cancelar-compra
Então a resposta deve ter o status code 200
E a quantidade em estoque do produto deve ser restaurada para X


```

9. Priorização da Execução dos Cenários de Teste

Prioridade	Cenários	Justificativa
Alta	Caminhos felizes de Usuários, Login, Produtos e **Carrinhos** (Criar, Concluir)	Valida os fluxos críticos e principais da API.
Média	Cenários negativos de todas as rotas	Garante o tratamento de erros e regras de negócio.
Baixa	Cenários de exceção e de robustez	Valida comportamentos menos comuns e resiliência.

10. Matriz de Risco

Risco Identificado	Impacto	Probabilidade	Plano de Contingência

Ambiente de testes instável ou fora do ar	Alto	Baixa	Comunicação constante com a equipe de infra/DevOps. Agendar testes em janelas de maior estabilidade.
Dados de teste inconsistentes	Médio	Medio	Criar scripts para limpar e popular o banco de dados antes de cada ciclo de teste, garantindo um estado conhecido.
Documentação Swagger desatualizada	Médio	Médio	Priorizar as User Stories e o comportamento real como fonte da verdade. Reportar as inconsistências como issues.

11. Cobertura de Testes

A cobertura de testes para a API ServeRest será medida através de múltiplas métricas para garantir não apenas que os requisitos funcionais sejam atendidos, mas também que a API seja robusta, segura e resiliente a entradas inesperadas. O objetivo é adotar uma abordagem estratégica que cubra a API sob diferentes perspectivas.

11.1 Cobertura por Requisitos (Regras de Negócio)

Esta é a métrica de maior prioridade. O objetivo é garantir **100% de cobertura** de todos os Critérios de Aceitação definidos nas User Stories (US001 a US004). Cada regra de negócio, como "Não é permitido excluir usuário com carrinho" ou "Rota exclusiva para administradores", será mapeada e validada por um ou mais cenários de teste. Os cenários BDD detalhados no Capítulo 8 são a implementação direta desta estratégia.

11.2 Cobertura por Endpoints e Métodos HTTP

Esta métrica garante que todas as rotas e verbos HTTP disponíveis na API sejam exercitados, conforme documentado no Swagger.

Rota	Método	Coberto
`/login`	`POST`	Sim
`/usuarios`	`GET`, `POST`	Sim
`/usuarios/{_id}`	`GET`, `PUT`, `DELETE`	Sim
`/produtos`	`GET`, `POST`	Sim
`/produtos/{_id}`	`GET`, `PUT`, `DELETE`	Sim
`/carrinhos`	`GET`, `POST`	Sim
`/carrinhos/{_id}`	`GET`	Sim
`/carrinhos/concluir-compra`	`DELETE`	Sim
`/carrinhos/cancelar-compra`	`DELETE`	Sim

11.3 Cobertura por Códigos de Status (Status Code)

O objetivo é validar que a API retorne os códigos de status HTTP corretos para cada situação, incluindo sucesso e, crucialmente, os diversos cenários de erro.

Código HTTP	Descrição	Endpoints de Exemplo	Coberto
`200 OK`	Sucesso em operações de busca ou alteração.	`GET /usuarios`, `PUT /produtos/{_id}`	Sim

`201 Created`	Sucesso na criação de um novo recurso.	`POST /usuarios`, `POST /produtos`	Sim
`400 Bad Request`	Erro do cliente (dados inválidos, regras de negócio violadas).	`POST /usuarios` (e-mail duplicado), `POST /carrinhos` (sem estoque)	Sim
`401 Unauthorized`	Erro de autenticação (token ausente, inválido ou expirado).	`POST /produtos` (sem token)	Sim
`403 Forbidden`	Erro de autorização (usuário não tem permissão).	`POST /produtos` (com usuário não-admin)	Sim

11.4 Cobertura por Corpo da Requisição (Payload)

Esta métrica foca na validação dos dados de entrada enviados no corpo das requisições `POST` e `PUT`. Utilizando as técnicas de Análise de Valor Limite e Particionamento de Equivalência, serão testadas diversas variações do payload para garantir a resiliência da API.

Exemplos de Cobertura:

- **Campo `password` em `/usuarios`:** Serão testados valores com menos de 5 caracteres, exatamente 5, entre 5 e 10, exatamente 10 e mais de 10 caracteres.
- **Campo `email` em `/usuarios`:** Serão testados e-mails com formato válido, inválido (`sem @`, `sem .com`), e com domínios proibidos (`gmail.com`, `hotmail.com`).
- **Payload de `/produtos`:** Serão enviadas requisições com o corpo completo, com campos obrigatórios ausentes e com campos contendo tipos de dados incorretos (ex: `preco` como string).

12. Testes Candidatos a Automação

A prioridade para implementação na collection Postman será:

1. **Smoke Test Suite (Prioridade Alta):** Cenários de "caminho feliz" para todas as rotas, incluindo criar usuário, fazer login, criar produto e **criar e finalizar um carrinho**.
2. **Regression Suite (Prioridade Média/Alta):** Todos os cenários negativos de validação de regras de negócio.

3. **Full Suite (Prioridade Média/Baixa):** Todos os cenários mapeados, incluindo os de robustez e exceção.

Esta automação será o núcleo da **Collection Postman** a ser entregue como parte final do challenge.