Por: João Pedro de Jesus Perin
Squad: 01 – Unit to Unity
Data: 06/02/2026

Versão do Plano:
1.0

Data de Criação:
02/02/2026

Data da Última Atualização:
06/02/2026

Responsável pela Elaboração:
João Pedro de Jesus Perin

Tempo Estimado de Execução:
Em torno de 16 horas (considerando o tempo integral do estágio durante a semana)

1. Escopo

1.1 Identificação do Projeto

Serão realizados testes exploratórios e funcionais na ServeRest API, uma API REST utilizada para simular um sistema de e-commerce, contemplando autenticação, usuários, produtos e carrinho de compras. O foco principal é validar a resiliência da API, a integridade dos dados, o tratamento de erros e o cumprimento do contrato HTTP.

1.2 Escopo do Teste

O escopo contempla os seguintes fluxos:

- Autenticação de usuários (Login)
- Cadastro, consulta, atualização e remoção de usuários
- Gerenciamento de produtos (operações administrativas)
- Criação, consulta e finalização/cancelamento de carrinho
- Validação de respostas HTTP e contratos

Fora do escopo:

- Testes de carga e performance
- Testes de segurança avançados (pentest)
- Interface gráfica (front-end)

2. Área Abrangida (Reprodução)

- Realizar requisições REST utilizando ferramentas como Postman
- Simular requisições válidas e inválidas
- Testar ausência, expiração e invalidez de token JWT

- Forçar erros de negócio (produto já existente, carrinho vazio, produto em carrinho)
- Validar códigos de status HTTP e mensagens retornadas

3. Mapeamento dos Endpoints

O mapeamento dos endpoints tem como objetivo identificar todas as rotas disponíveis na ServeRest API, seus métodos HTTP, finalidade, autenticação necessária e possíveis respostas esperadas. Esse mapeamento serve como base para a criação de cenários de teste funcionais, negativos e exploratórios.

3.1 Login

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/login	Autentica o usuário e retorna token JWT	Não	200 (Sucesso), 401 (Credenciais inválidas)

3.2 Usuário

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/usuario	Cadastro de novo usuário	Não	201 (Criado), 400 (Usuário já existente)
GET	/usuario	Lista usuários cadastrados	Não	200 (Sucesso)
GET	/usuario/{id}	Consulta usuário por ID	Não	200 (Encontrado), 400 (Não encontrado)
PUT	/usuario/{id}	Atualiza dados do usuário	Não	200 (Atualizado), 400 (Credencial duplicada)
DELETE	/usuario/{id}	Remove usuário	Não	200 (Deletado), 400 (Carrinho vinculado)

3.3 Produto

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/produto	Cria novo produto	Sim (ADM)	201 (Criado), 400 (Já existe), 401, 403
GET	/produto	Lista produtos	Não	200 (Sucesso)
GET	/produto/{id}	Consulta produto por ID	Não	200 (Encontrado), 400 (Não existe)
PUT	/produto/{id}	Atualiza produto	Sim (ADM)	200 (Atualizado), 400, 401, 403
DELETE	/produto/{id}	Remove produto	Sim (ADM)	200 (Deletado), 400 (Em carrinho), 401, 403

3.4 Carrinho

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/carrinho	Cria carrinho para o usuário	Sim	201 (Criado), 400 (Falha), 401
GET	/carrinho	Consulta carrinho do usuário	Sim	200 (Encontrado)
GET	/carrinho/{id}	Consulta carrinho por ID	Sim	200 (Encontrado), 400 (Não existe)
DELETE	/carrinho/concluir-compra	Finaliza compra	Sim	200 (Sucesso), 401

DELETE	/carrinho/ concluir-compra	Cancela compra	Sim	200 (Sucesso), 401

4. Ambiente de Testes

- Ferramenta de requisição: Postman
- Sistema Operacional: Windows 10/11
- API: ServeRest (ambiente público:)
- Protocolo: HTTP/HTTPS

5. Objetivos

5.1 Objetivo Geral

Explorar a ServeRest API por meio de testes funcionais e exploratórios, utilizando técnicas de sabotagem controlada para validar como a API reage a entradas inválidas, fluxos indevidos e falhas de autenticação, garantindo robustez e previsibilidade no consumo por aplicações clientes.

5.2 Objetivos Específicos

- Validar contratos de API (request/response)
- Garantir tratamento correto de erros HTTP (400, 401, 403, 404)
- Identificar riscos de inconsistência de dados
- Avaliar se a API responde de forma clara e padronizada a falhas

6. Riscos de Produto

Risco	Descrição	Impacto	Mitigação
Quebra de Autenticação	Acesso permitido sem token válido	Alto	Validação rigorosa de JWT em todos os endpoints protegidos
Inconsistência de Dados	Cadastro duplicado de usuários ou produtos	Alto	Validação rigorosa de JWT em todos os endpoints protegidos
Violação de Contrato	Campos ausentes ou tipos incorretos na resposta	Médio	Testes de contrato automatizados
Fluxo de Carrinho Inconsistente	Finalização ou exclusão indevida de carrinho	Médio	Validação de estado do carrinho antes das ações

7. Estratégia de Testes Exploratórios

Estratégia Escolhida: Testes Exploratórios Baseados em Sabotage Tours (API Sabotage)

Justificativa: A ServeRest API é amplamente utilizada como base para testes e estudos. A abordagem de Sabotage Tour permite explorar falhas além do caminho feliz, testando limites da

API, estados inválidos e sequências incorretas de chamadas, garantindo maior confiabilidade no consumo por aplicações externas.

8. Heurísticas Aplicadas (Adaptadas para API)

- Visibilidade do status do sistema: Retornos HTTP claros e consistentes
- Prevenção de erros: Bloqueio de dados inválidos no back-end
- Consistência: Padronização de mensagens e códigos de resposta
- Recuperação de erros: Mensagens claras ao consumidor da API

9.Cenários de Teste Funcionais

9.1 Cenários Positivos

- Realizar login com credenciais válidas e receber token JWT.
- Cadastrar usuário com dados válidos.
- Criar produto com token válido de administrador.
- Criar carrinho com produto existente.
- Concluir compra com carrinho válido.

9.2 Cenários Negativos

- Login com senha incorreta.
- Acesso a rota protegida sem token.
- Cadastro de usuário já existente.
- Criação de produto sem permissão de administrador.
- Exclusão de produto vinculado a carrinho.

9.3 Validação de Contrato

- Verificar obrigatoriedade de campos no request.
- Validar tipos de dados retornados no response.
- Conferir códigos HTTP conforme documentação (200, 201, 400, 401, 403, 404).

10.Tabela de Casos de Teste

10.1 Módulo: Login

ID	Cenário	Endpoint	Método	Pré-condição	Dados de Entrada	Resultado Esperado	Tipo
CT-LOGIN-01	Login com credenciais válidas	/login	POST	Usuário cadastrado	Email e senha válidos	Retornar token JWT e status 200	Positivo

CT-LOGIN-02	Login com senha inválida	/login	POST	Usuário cadastrado	Email válido e senha incorreta	Mensagem de erro e status 401	Negativo
CT-LOGIN-03	Login sem informar senha	/login	POST	Nenhuma	Email sem senha	Mensagem de campo obrigatório e status 400	Negativo
CT-LOGIN-04	Validar contrato do login	/login	POST	Usuário cadastrado	Credenciais válidas	Token no campo authorization	Contrato

10,**Módulo: Usuários**

ID	**Cenário**	**Endpoint**	**Método**	**Pré-condição**	**Dados de Entrada**	**Resultado Esperado**	**Tipo**
CT-USU-01	Criar usuário com dados válidos	/usuarios	POST	Nenhuma	Nome, email e senha válidos	Usuário criado com status 201	Positivo
CT-USU-02	Criar usuário com email duplicado	/usuarios	POST	Usuário já existente	Email já cadastrado	Mensagem de erro e status 400	Negativo
CT-USU-03	Consultar lista de usuários	/usuarios	GET	Nenhuma	N/A	Lista de usuários e status 200	Positivo
CT-USU-04	Validar contrato de usuário	/usuarios	GET	Nenhuma	N/A	Campos id, nome e email presentes	Contrato

10.3 Módulo: Produtos

ID	**Cenário**	**Endpoint**	**Método**	**Pré-condição**	**Dados de Entrada**	**Resultado Esperado**	**Tipo**
CT-PROD-01	Criar produto com token válido	/produtos	POST	Usuário autenticado	Nome, preço e quantidade	Produto criado com status 201	Positivo
CT-PROD-02	Criar produto sem token	/produtos	POST	Nenhuma	Dados válidos sem token	Erro de autenticação e status 401	Negativo
CT-PROD-03	Consultar produtos	/produtos	GET	Nenhuma	N/A	Lista de produtos e status 200	Positivo
CT-PROD-04	Validar contrato de	/produtos	GET	Nenhuma	N/A	Campos nome, preço	Contrato

	produto					e quantidade	

10.4 Módulo: Carrinho

ID	Cenário	Endpoint	Método	Pré-condição	Dados de Entrada	Resultado Esperado	Tipo
CT-CAR-01	Criar carrinho com produto válido	/carrinhos	POST	Usuário autenticado	Id do produto e quantidade	Carrinho criado e status 201	Positivo
CT-CAR-02	Criar carrinho sem autenticação	/carrinhos	POST	Nenhuma	Produto válido sem token	Erro de autenticação e status 401	Negativo
CT-CAR-03	Finalizar compra	/carrinhos /concluir-compra	DELETE	Carrinho existente	N/A	Compra finalizada e status 200	Positivo
CT-CAR-04	Validar contrato do carrinho	/carrinhos	GET	Usuário autenticado	N/A	Estrutura do carrinho válida	Contrato

11. Critérios de Entrada, Saída e Definição de Pronto

11.1 Critérios de Entrada

- API disponível e acessível.
- Documentação da ServeRest consultável.
- Ferramenta de testes configurada (Postman/Insomnia).

11.2 Critérios de Saída

- Todos os endpoints mapeados testados.
- Cenários críticos executados.
- Não existência de falhas bloqueadoras abertas.

11.3 Definição de Pronto (Definition of Done)

- Endpoints atendem ao contrato definido.
- Respostas retornam status HTTP corretos.
- Tratamento adequado de erros e exceções.

12. Evidências – Exemplos de Requisições (Modelo)

As evidências a seguir apresentam **modelos de requisições** baseadas na documentação Swagger oficial da ServeRest API. As requisições são ilustrativas e têm como objetivo demonstrar o preparo dos cenários de teste, não sendo necessária sua execução.

12.1 Login

12.1.1Login – POST /login

Cenário Positivo – Credenciais válidas

POST /login
Content-Type: application/json
{ **"email": "usuario@email.com",** **"password": "123456"** **}**

Resposta esperada: 200 OK + token JWT

Cenário Negativo – Senha inválida

POST /login
Content-Type: application/json
{ **"email": "usuario@email.com",** **"password": "123456"** **}**

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Campo email obrigatório (string)
- Campo password obrigatório (string)
- Resposta deve conter apenas o atributo authorization

12.2 Usuário

12.2.1Criar Usuário – POST /usuário

Criar Usuário – Cenário Positivo

POST /usuario
Content-Type: application/json
{ **"nome": "Usuário Teste",** **"email": "teste@email.com",** **"password": "123456",**

```
"administrador": "false"

}
```

Resposta esperada: 201 Created

Criar Usuário – Cenário Negativo (Usuário duplicado)

```
POST /usuario
Content-Type: application/json
{
"nome": "Usuário Teste",
"email": "teste@email.com",
"password": "123456",
"administrador": "false"
}
```

Resposta esperada: 400 Bad Request

12.2.2 Listar Usuários – GET /usuario

```
GET /usuario
```

Resposta esperada:200 OK

Cenário Negativo – Método inválido

```
POST /usuario
```

Resposta esperada: 405 Method Not Allowed

Validação de Contrato

- Retorno deve ser uma lista de usuários
- Cada usuário deve conter: id, nome, email, password, administrador
- Estrutura da resposta deve ser JSON

12.2.3 Consultar Usuário por ID – GET /usuario/{id}

Cenário Positivo – ID existente

```
GET /usuario/{id}
```

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

```
GET /usuario/id_invalido
```

Resposta esperada: 400 Bad Request

Validação de Contrato

- Parâmetro id é obrigatório
- Retorno deve conter os dados completos do usuário
- Estrutura JSON padronizada

12.2.4 Atualizar Usuário – PUT /usuario/{id}

Cenário Positivo – Atualização válida

PUT /usuario/{id}
Content-Type: application/json
{ **"nome": "Usuário Atualizado",** **"email": "novo@email.com",** **"password": "123456",** **"administrador": "false"** **}**

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

PUT /usuario/id_invalido

Resposta esperada: 400 Bad Request

Validação de Contrato

- id obrigatório
- Campos obrigatórios: nome, email, password, administrador
- Email deve ser válido
- administrador aceita apenas "true" ou "false"

12.2.5 Remover Usuário – DELETE /usuario/{id}

Cenário Positivo – Exclusão válida

DELETE /usuario/{id}

Resposta esperada: 200 OK

Cenário Negativo – Usuário inexistente

DELETE /usuario/id_invalido

Resposta esperada: 400 Bad Request

Validação de Contrato

- id obrigatório
- Usuário deve existir para ser removido
- Retorno deve confirmar a exclusão

12.3 Produtos

12.3.1 Criar Produto – POST /produto

Cenário Positivo – Produto válido

POST /produto
Authorization: Bearer <token_jwt>
Content-Type: application/json

{
"nome": "Produto Teste",
"preco": 100,
"descricao": "Produto para testes",
"quantidade": 10
}

Resposta esperada: 201 Created

Cenário Negativo – Sem autenticação

POST /produto

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Token JWT obrigatório

- Campos obrigatórios: nome, preco, descricao, quantidade
- preco e quantidade devem ser numéricos e positivos

12.3.2 Listar Produtos – GET /produto

Cenário Positivo – Listagem de produtos

```
GET /produto
```

Resposta esperada: 200 OK

Cenário Negativo – Endpoint inválido

```
GET /produtox
```

Resposta esperada: 404 Not Found

Validação de Contrato

- Retorno deve ser uma lista de produtos
- Estrutura JSON consistente

12.3.3 Consultar Produto por ID – GET /produto/{id}

Cenário Positivo – Produto existente

```
GET /produto/{id}
```

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

```
GET /produto/id_invalido
```

Resposta esperada: 400 Bad Request

Validação de Contrato

- Parâmetro id obrigatório
- Retorno deve conter os dados completos do produto

12.3.4 Atualizar Produto – PUT /produto/{id}

Cenário Positivo – Atualização válida

PUT /produto/{id}
Authorization: Bearer <token_jwt>
Content-Type: application/json

{
"nome": "Produto Atualizado",
"preco": 120,
"descricao": "Descrição atualizada",
"quantidade": 5
}

Resposta esperada: 200 OK

Cenário Negativo – Produto em carrinho

PUT /produto/{id}
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Validação de Contrato

- Token JWT obrigatório
- Produto não pode estar vinculado a um carrinho
- Campos devem respeitar os tipos definidos

12.3.5 Remover Produto – DELETE /produto/{id}

Cenário Positivo – Exclusão válida

DELETE /produto/{id}
Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Produto inexistente

DELETE /produto/id_invalido
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Validação de Contrato

- Token JWT obrigatório
- Produto deve existir para exclusão
- Retorno deve confirmar a remoção

12.4 Carrinho

12.4.1 Criar Carrinho – POST /carrinho

POST /carrinho
Authorization: Bearer <token_jwt>
Content-Type: application/json
{ **"produtos": [** **{** **"idProduto": "id_produto_exemplo",** **"quantidade": 1** **}** **]** **}**

Resposta esperada: 201 Created

Cenário Negativo – Produto inexistente

POST /carrinho
Authorization: Bearer <token_jwt>
Content-Type: application/json
{ **"produtos": [** **{** **"idProduto": "id_invalido",** **"quantidade": 1** **}** **]** **}**

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

POST /carrinho

Content-Type: application/json

```
{
  "produtos": [
    {
      "idProduto": "id_produto_exemplo",
      "quantidade": 1
    }
  ]
}
```

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Token JWT é obrigatório
- Campo produtos deve ser uma lista não vazia
- Campo idProduto deve ser um identificador válido
- Campo quantidade deve ser numérico e maior que zero
- A resposta deve conter mensagem de sucesso ou erro conforme o status HTTP

12.4.2 Consultar Carrinho do Usuário – GET /carrinho

Cenário Positivo – Carrinho existente

GET /carrinho

Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Usuário sem carrinho ativo

GET /carrinho

Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

GET /carrinho

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Token JWT é obrigatório**
- **Resposta deve conter os dados do carrinho**
- **Estrutura do JSON deve seguir o padrão definido no Swagger**

12.4.3 Consultar Carrinho por ID – GET /carrinho/{id}

Cenário Positivo – Carrinho existente

GET /carrinho/{id}
Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

GET /carrinho/id_invalido
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token inválido

GET /carrinho/{id}

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **ID deve ser válido**
- **Token JWT obrigatório**
- **Retorno deve conter lista de produtos e totais do carrinho**

12.4.4 Finalizar Compra – DELETE /carrinho/concluir-compra

Cenário Positivo – Carrinho válido

DELETE /carrinho/concluir-compra
Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Usuário sem carrinho ativo

DELETE /carrinho/concluir-compra
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

DELETE /carrinho/concluir-compra

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Carrinho deve existir e estar ativo**
- **Token JWT obrigatório**
- **Resposta deve conter mensagem de confirmação**

12.4.5 Cancelar Compra – DELETE /carrinho/cancelar-compra

Cenário Positivo – Carrinho válido

DELETE /carrinho/cancelar-compra
Authorization: Bearer <token_jwt>

Resposta esperada: 200 Sucesso

Cenário Negativo – Usuário sem carrinho ativo

DELETE /carrinho/cancelar-compra
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

DELETE /carrinho/cancelar-compra

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Token JWT obrigatório**
- **Carrinho deve estar ativo**
- **Resposta deve confirmar o cancelamento da operação**

13. Gerenciamento de Defeitos

Os defeitos serão registrados contendo:

- Endpoint afetado
- Método HTTP
- Payload enviado
- Resultado esperado vs obtido
- Evidências da resposta

14. Cronograma de Execução

- Dia 1: Login e Usuários
- Dia 2: Produtos
- Dia 3: Carrinho e retestes

15. Conclusão Técnica

Com base nos testes planejados, espera-se validar que a ServeRest API apresenta comportamento previsível, robusto e aderente aos contratos REST, sendo adequada para uso em ambientes de estudo, automação e integração de sistemas.