Documentação Técnica: API Usuarios Carrefour (Coleção Postman)

Este repositório contém dois arquivos principais para testes automatizados de APIs utilizando o Postman: `environment.json` e `collection.json`. Estes arquivos permitem a simulação, automação e validação de fluxos de autenticação, cadastro, consulta, atualização e exclusão de usuários no ambiente Carrefour, utilizando os endpoints do serviço serverest.dev.

Arquivo: `environment.json`

O arquivo `environment.json` define variáveis de ambiente do Postman. Ele permite armazenar valores dinâmicos usados em diferentes requisições, facilitando a reutilização e automação dos testes.

Principais Variáveis

Chave	Valor Padrão	Tipo	Descrição
url	https://serverest.dev	default	URL base para as requisições da API
token		default	Token de autenticação gerado no login
email		default	Email do usuário criado/atualizado
emailDuplicado		default	Email usado para testar duplicidade
randomEmail		any	Email gerado dinamicamente para testes
userFakeId		any	ID inexistente para testes negativos
_id		any	ID do usuário criado
usuarioDuplicado		default	Usuário duplicado para testes
userId		any	ID do usuário
loginEmail		any	Email utilizado para login
loginPassword		any	Senha utilizada para login
request_count		any	Contador de requisições (usado em teste de rate limit)

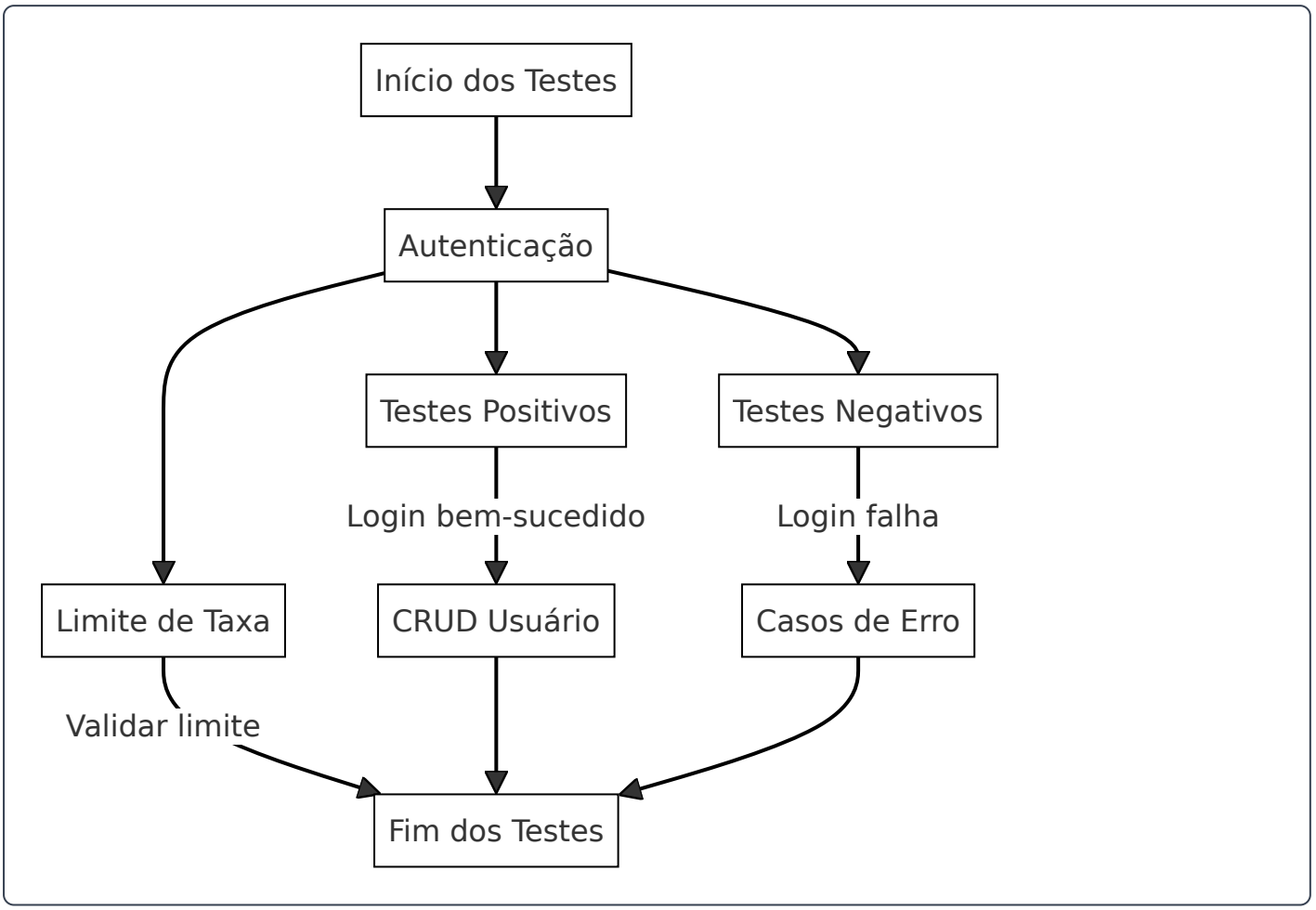
Arquivo: `collection.json`

Este arquivo define a coleção de requisições Postman para testar a API de usuários. A organização está dividida em quatro seções:

- **01. Autenticação**
- **02. Testes Positivos**
- **03. Testes Negativos**
- **04. Limite de Taxa**

Cada requisição contém scripts de pré-requisito e testes automáticos, validando tanto o funcionamento esperado quanto os cenários de erro.

Fluxo Geral das Requisições



Endpoints Disponíveis

A API utiliza o domínio `https://serverest.dev` para os seguintes endpoints de usuários:

Operação	Endpoint	Método
Login	`/login`	POST
Listar usuários	`/usuarios`	GET
Criar usuário	`/usuarios`	POST
Buscar usuário	`/usuarios/{_id}`	GET
Alterar usuário	`/usuarios/{_id}`	PUT
Excluir usuário	`/usuarios/{_id}`	DELETE

Detalhamento dos Endpoints e Exemplos de Uso

1. Autenticação

1.1 Login Válido (Cria usuário dinamicamente antes do login)

Fluxo:

- Pré-requisito: cria um usuário automaticamente.
- Realiza login com email e senha gerados.
- Armazena o token de autenticação no ambiente.

Exemplo de Requisição

```
POST /login
{
  "email": "user12345@teste.com",
  "password": "123456"
}
```

Exemplo de Resposta

```
{
  "message": "Login realizado com sucesso",
  "authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

Documentação Interativa

Login Válido

Login de Usuário

Export to Postman

Autentica um usuário válido e retorna um token JWT.

POST `https://serverest.dev/login`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "email": "user12345@teste.com",
  "password": "123456"
}
```

Code examples

```
curl -X POST "https://serverest.dev/login" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"email\": \"user12345@teste.com\",
  \"password\": \"123456\"
}'
```

Responses

200 400 401

Login realizado com sucesso.

```
{
  "message": "Login realizado com sucesso",
  "authorization": "Bearer <token>"
}
```

Campos obrigatórios não informados ou inválidos.

```
{
  "email": "email é obrigatório",
  "password": "password é obrigatório"
}
```

Email e/ou senha inválidos.

```
{
  "message": "Email e/ou senha inválidos"
}
```

1.2 Login Inválido - Senha Errada

Login Inválido - Senha Incorreta

Export to Postman

Tenta autenticar usuário com senha incorreta.

POST `https://serverest.dev/login`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "email": "admin@admin.com",
  "password": "odmin"
}
```

Code examples

```
curl -X POST "https://serverest.dev/login" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"email\": \"admin@admin.com\",
  \"password\": \"odmin\"
}'
```

Responses

400 401

Email e/ou senha inválidos.

```
{
  "message": "Email e/ou senha inválidos"
}
```

Email e/ou senha inválidos.

```
{
  "message": "Email e/ou senha inválidos"
}
```

1.3 Login Inválido - Usuário Inexistente

Login Inválido - Usuário Não Existe

Export to Postman

Tenta autenticar usuário inexistente.

POST `https://serverest.dev/login`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "email": "odmin@admin.com",
  "password": "admin"
}
```

Code examples

```
curl -X POST "https://serverest.dev/login" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"email\": \"odmin@admin.com\",
  \"password\": \"admin\"
}'
```

Responses

400 401

Email e/ou senha inválidos.

```
{
  "message": "Email e/ou senha inválidos"
}
```

Email e/ou senha inválidos.

```
{
  "message": "Email e/ou senha inválidos"
}
```

1.4 Login Inválido - Payload Vazio

Login Inválido - Payload Vazio

Export to Postman

Tenta autenticar sem informar email ou senha.

POST `https://serverest.dev/login`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{}
```

Code examples

```
curl -X POST "https://serverest.dev/login" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{}'
```

Responses

Campos obrigatórios não informados.

```
{
  "email": "email é obrigatório",
  "password": "password é obrigatório"
}
```

2. Testes Positivos

2.1 Validar Schema dos Usuários

Listar Usuários (Schema)

Export to Postman

Obtém lista de usuários e valida o schema de cada item.

GET `https://serverest.dev/usuarios`

Code examples

```
curl -X GET "https://serverest.dev/usuarios"
```

Responses

Lista de usuários recuperada com sucesso.

```
{
  "quantidade": 3,
  "usuarios": [
    {
      "nome": "Usuario Teste",
      "email": "user123@teste.com",
      "password": "123456",
      "administrador": "true",
      "_id": "123abc"
    }
  ]
}
```


2.2 Criar Usuário

Criar Usuário

Export to Postman

Cria um novo usuário administrado.

POST `https://serverest.dev/usuarios`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.


```
{
  "nome": "Usuario Teste",
  "email": "user123@teste.com",
  "password": "123456",
  "administrador": "true"
}
```


Code examples


```
curl -X POST "https://serverest.dev/usuarios" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"Usuario Teste\",
  \"email\": \"user123@teste.com\",
  \"password\": \"123456\",
  \"administrador\": \"true\"
}'
```

Responses

201 400

Usuário criado com sucesso.


```
{
  "message": "Cadastro realizado com sucesso",
  "_id": "5f6f9a8f6c3f4a0017b8f8b7"
}
```


Erro de validação nos campos obrigatórios.


```
{
  "email": "email já está sendo usado"
}
```


2.3 Listar Usuários

Listar Usuários

Export to Postman

Retorna todos os usuários cadastrados.

GET `https://serverest.dev/usuarios`

Code examples

```
curl -X GET "https://serverest.dev/usuarios"
```

Responses

Lista de usuários retornada com sucesso.


```
{
  "quantidade": 3,
  "usuarios": [{...}]
}
```


2.4 Buscar Usuário por ID

Buscar Usuário por ID

Export to Postman

Retorna o usuário correspondente ao ID informado.

GET `https://serverest.dev/usuarios/{{_id}}`

Path parameters

`_id` string • path required

ID do usuário

Code examples

```
curl -X GET "https://serverest.dev/usuarios/{{_id}}"
```

Responses

200 400

Usuário encontrado com sucesso.

```
{
  "nome": "Usuario Teste",
  "email": "user123@teste.com",
  ...
}
```

Usuário não encontrado.

```
{
  "message": "Usuário não encontrado"
}
```

2.5 Alterar Usuário

Alterar Usuário

Export to Postman

Atualiza os dados do usuário pelo ID.

PUT `https://serverest.dev/usuarios/{{_id}}`

Headers

`Content-Type` string • header required

application/json

Path parameters

`_id` string • path required

ID do usuário

Request body

JSON payload required for this request.

```
{
  "nome": "Usuario Atualizado",
  "email": "user123@teste.com",
  "password": "123456",
  "administrador": "false"
}
```

Code examples

```
curl -X PUT "https://serverest.dev/usuarios/{{_id}}" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"Usuario Atualizado\",
  \"email\": \"user123@teste.com\",
  \"password\": \"123456\",
  \"administrador\": \"false\"
}'
```

Responses

200 400

Usuário alterado com sucesso.

```
{
  "message": "Registro alterado com sucesso"
}
```

Erro ao alterar usuário.

```
{
  "message": "Usuário não encontrado"
}
```

2.6 Excluir Usuário

Excluir Usuário

Export to Postman

Remove o usuário do sistema pelo ID.

DELETE `https://serverest.dev/usuarios/{{_id}}`

Path parameters

`_id` string • path required

ID do usuário

Code examples

```
curl -X DELETE "https://serverest.dev/usuarios/{{_id}}"
```

Responses

200 400

Exclusão realizada com sucesso.

```
{
  "message": "Registro excluído com sucesso"
}
```

Usuário não encontrado para exclusão.

```
{
  "message": "Usuário não encontrado para exclusão"
}
```

3. Testes Negativos

Os testes negativos cobrem cenários de falha como duplicidade de email, campos obrigatórios não preenchidos, dados inválidos e operações sobre usuários inexistentes.

3.1 Criar Usuário Duplicado

Criar Usuário Duplicado

Export to Postman

Tenta cadastrar um usuário com email já existente.

POST `https://serverest.dev/usuarios`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "nome": "Usuario Duplicado",
  "email": "fulano@qa.com",
  "password": "123456",
  "administrador": "true"
}
```

Code examples

```
curl -X POST "https://serverest.dev/usuarios" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"Usuario Duplicado\",
  \"email\": \"fulano@qa.com\",
  \"password\": \"123456\",
  \"administrador\": \"true\"
}'
```

Responses

Email já está sendo usado.

```
{
  "message": "Este email já está sendo usado"
}
```

3.2 Consultar Usuário Inexistente

Consultar Usuário Inexistente

Export to Postman

Tenta buscar um usuário que não existe.

GET `https://serverest.dev/usuarios/{{userFakeId}}`

Path parameters

`userFakeId` string • path required

ID inexistente do usuário

Code examples

```
curl -X GET "https://serverest.dev/usuarios/{{userFakeId}}"
```

Responses

400 404

ID inválido.

```
{
  "message": "ID inválido"
}
```

Usuário não encontrado.

```
{
  "message": "Usuário não encontrado"
}
```

3.3 Atualizar Usuário Inexistente

Atualizar Usuário Inexistente

Export to Postman

Tenta atualizar dados de um usuário que não existe.

PUT `https://serverest.dev/usuarios/{{userFakeId}}`

Headers

Content-Type string • header required

application/json

Path parameters

userFakeId string • path required

ID inexistente do usuário

Request body

JSON payload required for this request.

```
{
  "nome": "Usuario Teste",
  "email": "email@teste.com",
  "password": "123456",
  "administrador": "true"
}
```

Code examples

```
curl -X PUT "https://serverest.dev/usuarios/{{userFakeId}}" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"Usuario Teste\",
  \"email\": \"email@teste.com\",
  \"password\": \"123456\",
  \"administrador\": \"true\"
}'
```

Responses

400 404

ID inválido.

```
{
  "message": "ID inválido"
}
```

Usuário não encontrado.

```
{
  "message": "Usuário não encontrado"
}
```

3.4 Deletar Usuário Inexistente

Deletar Usuário Inexistente

Export to Postman

Tenta deletar um usuário que não existe.

DELETE `https://serverest.dev/usuarios/{{userFakeId}}`

Path parameters

`userFakeId` string • path required

ID inexistente do usuário

Code examples

```
curl -X DELETE "https://serverest.dev/usuarios/{{userFakeId}}"
```

Responses

200 400

Nenhum registro excluído.

```
{
  "message": "Nenhum registro excluído"
}
```

Usuário não encontrado para exclusão.

```
{
  "message": "Usuário não encontrado para exclusão"
}
```

3.5-3.9 Validação de Campos Obrigatórios

Cada campo obrigatório (nome, email, senha, administrador) é testado individualmente e em conjunto, para garantir que a API retorna status 400 e mensagens apropriadas quando esses campos não são informados ou preenchidos incorretamente.

3.10 Criar Usuário com Email Inválido

Criar Usuário - Email Inválido

Export to Postman

Tenta cadastrar usuário com formato de email inválido.

POST `https://serverest.dev/usuarios`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "nome": "Usuario Email Invalido",
  "email": "email-invalido",
  "password": "123456",
  "administrador": "true"
}
```

Code examples

```
curl -X POST "https://serverest.dev/usuarios" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"Usuario Email Invalido\",
  \"email\": \"email-invalido\",
  \"password\": \"123456\",
  \"administrador\": \"true\"
}'
```

Responses

Formato de email inválido.

```
{
  "email": "email inválido"
}
```

3.11 Criar Usuário com Dados Inválidos

Criar Usuário - Dados Inválidos

Export to Postman

Tenta cadastrar usuário com dados fora do esperado.

POST `https://serverest.dev/usuarios`

Headers

`Content-Type` string • header required

application/json

Request body

JSON payload required for this request.

```
{
  "nome": "123",
  "email": "true",
  "password": [],
  "administrador": "999"
}
```

Code examples

```
curl -X POST "https://serverest.dev/usuarios" \
  -H "Content-Type: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  \"nome\": \"123\",
  \"email\": \"true\",
  \"password\": [],
  \"administrador\": \"999\"
}'
```

Responses

Erro de validação.

```
{
  "message": "Dados inválidos"
}
```

3.12 Usuário sem Token

Listar Usuários sem Token

Export to Postman

Tenta listar usuários sem enviar token de autenticação.

GET `https://serverest.dev/usuarios`

Code examples

```
curl -X GET "https://serverest.dev/usuarios"
```

Responses

Resposta pode ser retornada sem autenticação (API de exemplo).

```
{
  "quantidade": 3,
  "usuarios": [{...}]
}
```

4. Limite de Taxa

O endpoint de usuários é chamado repetidamente para simular e analisar o comportamento de rate limit, armazenando o histórico em variáveis de ambiente.

4.1 Teste de Limite de Requisições

Teste de Limite de Taxa

Export to Postman

Executa múltiplas requisições para validar comportamento de rate limit.

`GET` `https://serverest.dev/usuarios`

Headers

`Authorization` string • header required

Bearer {{token}}

Code examples

```
curl -X GET "https://serverest.dev/usuarios" \
  -H "Authorization: Bearer {{token}}"
```

Responses

200 400 429

Requisição aceita.

```
{
  "quantidade": 3,
  "usuarios": [{...}]
}
```

Limite atingido ou erro de validação.

```
{
  "message": "Limite excedido"
}
```

Too Many Requests.

```
{
  "message": "Too Many Requests"
}
```

Exemplos de Uso na Prática

Criar Novo Usuário com Email Dinâmico

```
curl -X POST https://serverest.dev/usuarios \
-H "Content-Type: application/json" \
-d '{
  "nome": "Usuario Teste",
  "email": "user32452@teste.com",
  "password": "123456",
  "administrador": "true"
}'
```

Autenticar e Utilizar o Token

```
curl -X POST https://serverest.dev/login \
-H "Content-Type: application/json" \
-d '{
  "email": "user32452@teste.com",
  "password": "123456"
}'
# Retire o token da resposta e utilize nos headers das próximas requisições
```

Buscar Usuário por ID

```
curl -X GET https://serverest.dev/usuarios/5f6f9a8f6c3f4a0017b8f8b7 \
-H "Authorization: Bearer <token>"
```

Scripts e Validações Automatizadas

- **Pré-requisitos**: Geram dados dinâmicos, criam usuários e preparam o ambiente de teste.
- **Testes**: Validam status code, mensagens de erro, existência de propriedades, tempo de resposta, formato JSON, e logs de debug.
- **Histórico de Rate Limit**: Armazena quantidade de requisições e tempos de resposta para análise posterior.

Ambiente Dinâmico
Os testes aproveitam variáveis de ambiente para automação, geração de dados dinâmicos e controle de fluxo completo.

Cobertura Completa
A coleção cobre todos os cenários: sucesso, erro, validação de obrigatórios, duplicidade e limites de requisição.

Resumo

Esta documentação detalha o funcionamento e uso da coleção de testes automatizados para a API de usuários Carrefour, incluindo todos os endpoints, exemplos de entrada e saída, scripts de automação e validações. Use este material para entender, executar e validar integrações com a API de maneira rápida, segura e automatizada!