Raphael - Challenge ServeRest

Plano de Teste

Apresentação

Este documento descreve a estratégia de testes para a API ServeRest, uma aplicação que simula um ambiente de e-commerce para fins de treinamento e validação de testes. A estrutura deste plano abrange o objetivo, o contexto do sistema e a abordagem de testes a ser seguida.

Objetivo

Garantir a qualidade do e-commerce ServeRest por meio de testes que validem o funcionamento correto dos endpoints e assegurem a segurança da aplicação.

Resumo

O ServeRest é uma plataforma de e-commerce genérica que contempla três tipos de usuários: administrador, administrador do sistema e usuário comum - público-alvo da plataforma. Este planejamento de testes será executado em resposta a problemas identificados nas ferramentas do sistema.

A hipótese central deste teste é que todos os endpoints estejam corretamente implementados e funcionais, permitindo a validação completa do fluxo de operações do e-commerce.

Pessoas Envolvidas:

- Raphael Sousa Rabelo Rates

Escopo

Nessa seção, é explicada como as quais funcionalidades, módulos ou áreas do sistema serão cobertas pelos testes, evitando ambiguidades e garantindo foco. Mostraremos as funcionalidades principais da aplicação (ex.: cadastro de usuário, login, carrinho de compras, processamento de pedidos), tipos de testes planejados (unitários, integração, API, performance, segurança), ambiente de testes (desenvolvimento, homologação, produção simulada).

EndPoints de teste

- Login
  - `POST /login`

Usuários

- `GET /usuarios`
- `POST /usuarios`
- `GET /usuarios/{_id}`
- `DELETE /usuarios/{_id}`
- `PUT /usuarios/{_id}`

Produtos

- `GET /produtos`
- `POST /produtos`
- `GET /produtos/{_id}`
- `DELETE /produtos/{_id}`
- `PUT /produtos/{_id}`

Carrinhos

- `GET /carrinhos`
- `POST /carrinhos`
- `GET /carrinhos/{_id}`
- `DELETE /carrinhos/concluir-compra`
- `DELETE /carrinhos/cancelar-compra`

Todas as rotas foram submetidas aos seguintes tipos de testes: Unitários, Desempenho e Regressivo)

Testes Unitários

Os testes serão realizados com base na validação do retorno dos dados esperados para cada regra de negócio em cada endpoint da aplicação. As regras de negócio, incluindo aspectos técnicos detalhados, estão documentadas no registro Swagger da aplicação.

Desempenho

Os testes verificam o desempenho de cada endpoint com base no tempo de resposta da da request.

Regressão

Teste regressivo que executa todas as rotas em apenas um Run, verificando os fluxos do sistema para verificar se não tem nenhuma rota com bugs. É utilizado a automação do Postman usando Flow, ou mesmo utilizando uma automação mais robusta com um framework de testes como RobotFramework

Análise

Existe algumas URLs que possuem alto riscos caso obtenham algum tipo de bug, sendo elas as seguintes abaixo:

- `POST /login`
- `POST /usuarios`
- `POST /produtos`
- `DELETE /carrinhos/concluir-compra`

Técnicas Aplicadas

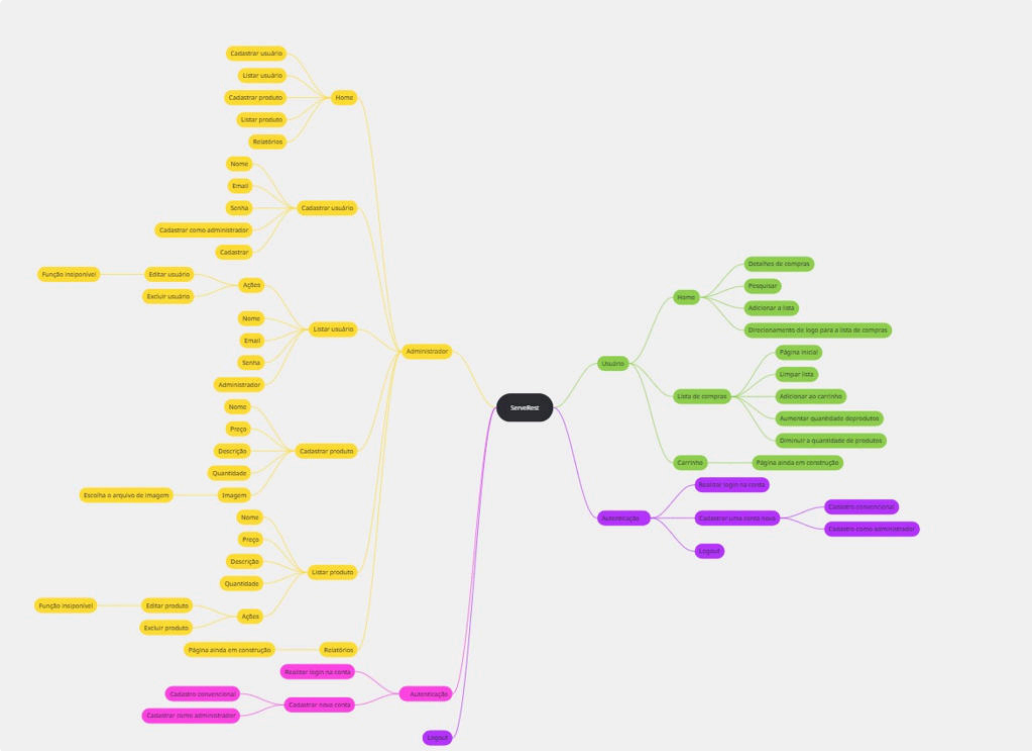
- Testes de API: usando scripts do postman para verificar as regras de negocio de cada rota e/ou testes unitários em Robot-framework.
- Testes de automação: Usando flows automatizados com Monitores para verificação automática e a utilização do framework para testes automatizados Robot-Framework.

Tipos de Priorização de Testes

Uma forma de caracterizar os tipos de testes que devem ser realizados.

- Fácil
- Médio (Relevante)
- Difícil (Complicada)
- Muito difícil (Urgência)

Mapa Mental da Aplicação



Cenários de Teste Planejados

Priorização da execução dos cenários de teste

Cenário	Prioridade	Justificativa
Criar usuário com dados válidos (POST `/usuarios`)	Alta	Fluxo principal de cadastro de usuário; falha compromete a gestão de usuários.
Criar usuário com dados inválidos (POST `/usuarios`)	Alta	Garante que a API valida corretamente entradas inválidas; falha pode permitir usuários inconsistentes.
Editar usuário existente (PUT `/usuarios/{_id}`)	Média	Operação de manutenção importante, mas não impede o uso do sistema.

Buscar usuário por ID válido (GET `/usuarios/{_id}`)	Alta	Fluxo essencial para consultas; falha impacta funcionalidade básica.
Buscar usuário por ID inválido (GET `/usuarios/{_id}`)	Média	Cenário de exceção; importante para feedback adequado, mas não crítico.
Excluir usuário existente (DELETE `/usuarios/{_id}`)	Alta	Essencial para controle de usuários; falha compromete segurança e gerenciamento.
Criar produto com autenticação válida (POST `/produtos`)	Alta	Fluxo principal de cadastro de produto; falha compromete todo o uso da API de produtos.
Criar produto com autenticação inválida (POST `/produtos`)	Alta	Garante que a API valida corretamente os dados de entrada; falha aqui pode permitir produtos inválidos.
Atualizar produto existente (PUT `/produtos/{_id}`)	Média	Caso de manutenção; importante mas não bloqueia cadastro e listagem.
Atualizar produto inexistente (PUT `/produtos/{_id}`)	Média	Cenário de exceção; útil para verificar comportamento, mas não crítico.

Buscar produto por ID válido (GET `/produtos/{_id}`)	Alta	Essencial para consultas; falha impacta funcionalidade básica.
Buscar produto por ID inválido (GET `/produtos/{_id}`)	Média	Cenário de exceção; importante para feedback, mas não crítico.
Excluir produto existente (DELETE `/produtos/{_id}`)	Alta	Essencial para manutenção de estoque e segurança.
Criar carrinho (POST `/carrinhos`)	Alta	Fluxo principal de compra; falha impede uso do sistema de carrinhos.
Listar carrinhos (GET `/carrinhos`)	Média	Importante para monitoramento, mas não crítico para criar/editar carrinho.
Buscar carrinho por ID (GET `/carrinhos/{_id}`)	Média	Útil para validação e monitoramento; falha não impede criação.
Concluir compra (DELETE `/carrinhos/concluir-compra`)	Alta	Etapa final da compra; falha bloqueia operação de checkout.
Cancelar compra (DELETE	Alta	Garante que produtos retornem ao estoque corretamente; falha pode gerar

`/carrinhos/cancelar-compra`)		inconsistência de estoque.

Matriz de Riscos

Rota	Método	Cenário	Impacto	Probabilidade	Classificação
/usuarios	POST	Cadastro com dados inválidos	Alto	Alta	Crítico
/usuarios	POST	Cadastro com email duplicado	Médio	Alta	Alto
/usuarios	GET	Listagem sem autenticação	Baixo	Alta	Médio
/usuarios/{id}	GET	Busca de usuário inexistente	Baixo	Média	Baixo
/usuarios/{id}	DELETE	Exclusão sem permissão	Alto	Média	Alto
/usuarios/{id}	PUT	Edição com dados inválidos	Médio	Alta	Alto
/produtos	POST	Criação sem autenticação	Alto	Alta	Crítico

/produtos	POST	Criação com dados inválidos	Alto	Alta	Crítico
/produtos	GET	Listagem com filtros maliciosos	Médio	Média	Médio
/produtos/{id}	GET	Busca de produto inexistente	Baixo	Média	Baixo
/produtos/{id}	DELETE	Exclusão sem permissão	Alto	Média	Alto
/produtos/{id}	PUT	Edição que quebra integridade	Alto	Média	Alto
/carrinhos	POST	Criação com produtos inexistentes	Médio	Alta	Alto
/carrinhos	POST	Criação com quantidades inválidas	Médio	Alta	Alto
/carrinhos	GET	Listagem expõe dados sensíveis	Alto	Média	Alto

/carrinhos/{id}	GET	Busca de carrinho de outro usuário	Alto	Média	Alto
/carrinhos/concluir-compra	DELETE	Conclusão sem itens no carrinho	Médio	Média	Médio

Cobertura de testes

Cenário de Teste	Endpoint	Método	Cobertura de Teste	Automação no Postman	Dados Utilizados	Validações Principais
Criar produto com autenticação válida	/produtos	POST	Cadastro bem-sucedido com dados válidos	Script de pré-request com dados aleatórios	Nome, preço, descrição, quantidade aleatórios	Status 201, estrutura JSON correta, dados correspondentes
Criar produto com dados inválidos	/produtos	POST	Validação de campos obrigatórios e regras	Script com múltiplos cenários de dados inválidos	Campos vazios, valores negativos, tipos incorretos	Status 400, mensagens de erro específicas
Atualizar produto inexistente	/produtos/{id}	PUT	Verificação de ID inexistente	Geração de IDs aleatórios não	ID aleatório não cadastrado,	Status 404, não criação de novo recurso

				existentes	dados de atualização	
Listar produtos	/produtos	GET	Recuperação de lista completa	Parâmetros de query opcionais	-	Status 200, array não vazio, estrutura correta
Buscar produto por ID existente	/produtos/{id}	GET	Recuperação de produto específico	Uso de ID de produto previamente criado	ID válido de produto existente	Status 200, dados completos do produto
Buscar produto por ID inexistente	/produtos/{id}	GET	Tratamento de ID não encontrado	Geração de IDs aleatórios não existentes	ID aleatório não cadastrado	Status 404, mensagem de não encontrado
Deletar produto com autenticação	/produtos/{id}	DELETE	Remoção de produto existente	Uso de ID de produto previamente criado	ID válido de produto existente	Status 200, mensagem de confirmação
Deletar produto inexistente	/produtos/{id}	DELETE	Tratamento de ID não encontrado	Geração de IDs aleatórios não	ID aleatório não cadastrado	Status 404, mensagem de não

				existentes		encontrado
Editar produto com dados parciais	/produtos/{id}	PUT	Atualização parcial de campos	Uso de ID existente com campos específicos	ID válido, apenas alguns campos atualizados	Status 200, apenas campos enviados atualizados
Criar carrinho com produto válido	`/carrinhos`	POST	Cadastro de carrinho com item válido	Script de pré-request com `idProduto` válido	`idProduto`, `quantidade: 1`	Status **201**, estrutura JSON correta, vínculo com usuário
Criar múltiplos carrinhos para o mesmo usuário	`/carrinhos`	POST	Validação de regra de negócio	Script repetindo criação de carrinho com mesmo token	`idProduto`, `quantidade: 1`	Esperado **400** (já existe carrinho ativo), Obtido criação duplicada
Adicionar produto inexistente ao carrinho	`/carrinhos`	POST	Validação de integridade de dados	Script com `idProduto` inexistente	`idProduto: invalido123`, `quantidade: 1`	Esperado **404**, obtido criação com produto inválido

Adicionar produto com quantidade inválida	`/carrinhos`	POST	Teste de limite	Script com valores extremos de quantidade	`quantidade: 0`, `quantidade: -5`, `quantidade: 1`	Status **400** para inválidos, **201** para válido
Finalizar compra com carrinho vazio	`/carrinhos/concluir-compra`	POST	Regra de negócio (fluxo de compra)	Script executando compra sem itens	Token válido, carrinho vazio	Esperado **400**, obtido **200** (compra concluída)
Excluir carrinho do próprio usuário	`/carrinhos/{idCarrinho}`	DELETE	Exclusão bem-sucedida	Script com `idCarrinho` do usuário logado	`idCarrinho` válido	Status **200**, carrinho removido
Excluir carrinho de outro usuário	`/carrinhos/{idCarrinho}`	DELETE	Validação de segurança	Script com token de usuário diferente	`idCarrinho` de outro usuário	Esperado **403**, obtido exclusão bem-sucedida
Criar carrinho com	`/carrinhos`	POST	Script de pré-request para	Status **201**, resposta JSON	Criar carrinho com	`/carrinhos`

produto válido			gerar `idProduto` válido e validações no **pm.test**	com campo `_id`	produto válido	
Teste de Limite: tentar criar carrinho sem produtos (`produtos: []`)	`/carrinhos`	POST	Script enviando array vazio de produtos	Esperado **400**, mensagem de erro clara	**Teste de Limite:** tentar criar carrinho sem produtos (`produtos: []`)	`/carrinhos`
Adicionar produto com quantidade inválida	`/carrinhos`	POST	Runner variando `quantidade = 0, -1`, validações no **pm.test**	Status **400**, mensagem de erro padronizada	Adicionar produto com quantidade inválida	`/carrinhos`
Teste de Limite: adicionar produto com `quantidade`	`/carrinhos`	POST	Script com quantidade mínima	Esperado **201**, carrinho criado com sucesso	**Teste de Limite:** adicionar produto com `quantidade`	`/carrinhos`

`= 1` (mínimo válido)					`= 1` (mínimo válido)	

Testes candidatos a automação

Cenário	Endpoint	Justificativa para automação
Criar produto com autenticação válida (POST)	`/produtos`	Fluxo principal, precisa ser verificado sempre
Criar produto com dados inválidos (POST)	`/produtos`	Teste repetitivo e previsível, garante consistência
Atualizar produto inexistente (PUT cria novo)	`/produtos/{id}`	Pode ser automatizado para validar exceções recorrentes
Criar carrinho com produto válido	`/carrinhos`	Cenário feliz (happy path), fácil de automatizar com dados randômicos.
Adicionar produto com quantidade inválida	`/carrinhos`	Teste de limite clássico, fácil de parametrizar (0, -1, -5).

Testes de automação

Por Rota

Foram feitos testes para cada rota, no documento só serão mostrados as rotas com seus devidos métodos mais comuns na aplicação

1. /login

```
pm.test("Status 200 - Login realizado com sucesso", function () {
    pm.expect(pm.response.code).to.eql(200);
});

pm.test("Mensagem de login bem-sucedido", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("message");
    pm.expect(jsonData.message).to.eql("Login realizado com sucesso");
});

pm.test("Token de autorização é retornado", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("authorization");
    pm.expect(jsonData.authorization).to.be.a('string');
    pm.expect(jsonData.authorization).to.not.be.empty;
    pm.expect(jsonData.authorization).to.include("Bearer ");
    pm.environment.set("auth_token", jsonData.authorization);
    console.log("Token obtido:", jsonData.authorization.substring(0, 50) + "...");
});

pm.test("Response time é aceitável", function () {
    pm.expect(pm.response.responseTime).to.be.below(2000);
});

pm.test("Token tem formato válido", function () {
    var jsonData = pm.response.json();
    const token = jsonData.authorization;
    pm.expect(token.length).to.be.at.least(100);
    pm.expect(token).to.include(".");
});


pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- Login
  - `POST /login`

```
pm.test("Status 200 - Login realizado com sucesso", function () {
    pm.expect(pm.response.code).to.eql(200);
});

pm.test("Mensagem de login bem-sucedido", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("message");
    pm.expect(jsonData.message).to.eql("Login realizado com sucesso");
});

pm.test("Token de autorização é retornado", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("authorization");
    pm.expect(jsonData.authorization).to.be.a('string');
    pm.expect(jsonData.authorization).to.not.be.empty;
    pm.expect(jsonData.authorization).to.include("Bearer ");
    pm.environment.set("auth_token", jsonData.authorization);
    console.log("Token obtido:", jsonData.authorization.substring(0, 50) + "...");
});

pm.test("Response time é aceitável", function () {
    pm.expect(pm.response.responseTime).to.be.below(2000);
});

pm.test("Token tem formato válido", function () {
    var jsonData = pm.response.json();
    const token = jsonData.authorization;
    pm.expect(token.length).to.be.at.least(100);
    pm.expect(token).to.include(".");
});


pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- **Usuários**
  - `GET /usuarios`

```
pm.test("Status code é 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Resposta tem lista de usuários", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("usuarios");
    pm.expect(jsonData.usuarios).to.be.an("array");
    pm.expect(jsonData).to.have.property("quantidade");
});

pm.test("Usuário(s) retornado(s) tem campos esperados", function () {
    var user = pm.response.json().usuarios[0];
    pm.expect(user).to.have.property("nome");
    pm.expect(user).to.have.property("email");
    pm.expect(user).to.have.property("administrador");
    pm.expect(user).to.have.property("password");
    pm.expect(user).to.have.property("_id");
});
pm.test("Response time is less than 1000ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

- `POST /usuarios`

```
if (pm.response.code === 201) {
    pm.test("Status 201 - Cadastro realizado com sucesso", function () {
        pm.expect(pm.response.code).to.eql(201);
    });
    pm.test("Resposta contém mensagem de sucesso", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
    });
    pm.test("Resposta contém ID do usuário criado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("_id");
        pm.expect(jsonData._id).to.be.a('string');
        pm.expect(jsonData._id).to.not.be.empty;
        pm.environment.set("user_id", jsonData._id);
        console.log("ID do usuário criado:", jsonData._id);
    });
    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 400) {
    pm.test("Status 400 - Email já está em uso", function () {
        pm.expect(pm.response.code).to.eql(400);
    });

    pm.test("Mensagem de email já cadastrado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Este email já está sendo usado");
    });

    pm.test("Resposta não contém ID em caso de erro", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.not.have.property("_id");
    });
}
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

- `GET /usuarios/{_id}`

```
pm.test("Status code é 200 ou 400", function () {
    pm.expect([200, 400]).to.include(pm.response.code);
});

if (pm.response.code === 200) {
    pm.test("Resposta contém campos esperados", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("nome");
        pm.expect(jsonData).to.have.property("email");
        pm.expect(jsonData).to.have.property("administrador");
        pm.expect(jsonData).to.have.property("_id");
    });
}

if (pm.response.code === 400) {
    pm.test("Mensagem de usuário não encontrado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData.message).to.eql("Usuário não encontrado");
    });
}
pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

- `DELETE /usuarios/{_id}`

```
// Test Script - Executa DEPOIS da requisição DELETE
console.log("Status code recebido:", pm.response.code);
console.log("Resposta:", pm.response.text());

// Teste para status 200 - Exclusão bem-sucedida ou nenhum registro excluído
if (pm.response.code === 200) {
    pm.test("Status 200 - Operação concluída", function () {
        pm.expect(pm.response.code).to.eql(200);
    });

    pm.test("Resposta contém mensagem apropriada", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");

        // Pode retornar duas mensagens diferentes no status 200
        const possibleMessages = [
            "Registro excluído com sucesso",
            "Nenhum registro excluído",
            "Não é permitido excluir usuário com carrinho cadastrado"
        ];

        pm.expect(possibleMessages).to.include(jsonData.message);
    });

    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}

// Teste para status 400 - Usuário com carrinho cadastrado
if (pm.response.code === 400) {
    pm.test("Status 400 - Não é permitido excluir usuário com carrinho", function () {
        pm.expect(pm.response.code).to.eql(400);
    });

    pm.test("Mensagem de erro correta", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Não é permitido excluir usuário com carrinho cadastrado");
    });

    pm.test("Resposta contém ID do carrinho", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("idCarrinho");
        pm.expect(jsonData.idCarrinho).to.be.a('string');
        pm.expect(jsonData.idCarrinho).to.not.be.empty;

        // Salva o ID do carrinho para possível uso em outros testes
        pm.environment.set("carrinho_id_bloqueio", jsonData.idCarrinho);
        console.log("ID do carrinho que impede exclusão:", jsonData.idCarrinho);
    });
}
```

- `PUT /usuarios/{_id}`

```
if (pm.response.code === 200) {
    pm.test("Status 200 - Alteração realizada com sucesso", function () {
        pm.expect(pm.response.code).to.eql(200);
    });

    pm.test("Mensagem de alteração bem-sucedida", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Registro alterado com sucesso");
    });

    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}

// Teste para status 201 - Cadastro realizado (quando usuário não existe)
if (pm.response.code === 201) {
    pm.test("Status 201 - Cadastro realizado com sucesso", function () {
        pm.expect(pm.response.code).to.eql(201);
    });

    pm.test("Mensagem de cadastro bem-sucedido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
    });

    pm.test("ID do novo usuário é retornado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("_id");
        pm.expect(jsonData._id).to.be.a('string');
        pm.expect(jsonData._id).to.not.be.empty;

        // Salva o ID para uso em outros testes
        pm.environment.set("novo_usuario_id", jsonData._id);
        console.log("ID do novo usuário criado:", jsonData._id);
    });
}

// Teste para status 400 - Email já cadastrado
if (pm.response.code === 400) {
    pm.test("Status 400 - Email já está em uso", function () {
        pm.expect(pm.response.code).to.eql(400);
    });

    pm.test("Mensagem de email já cadastrado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Este email já está sendo usado");
    });

    pm.test("Resposta não contém ID em caso de erro", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.not.have.property("_id");
    });
}

// Testes que se aplicam a todos os casos
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- **Produtos**
  - `GET /produtos`

```
pm.test("Status 200 - Lista de produtos retornada", function () {
    pm.expect(pm.response.code).to.eql(200);
});
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});
pm.test("Estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
pm.test("Response time is less than 1000ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
pm.test("Resposta contém estrutura correta", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("quantidade");
    pm.expect(jsonData.quantidade).to.be.a('number');
    pm.expect(jsonData.quantidade).to.be.at.least(0);
    pm.expect(jsonData).to.have.property("produtos");
    pm.expect(jsonData.produtos).to.be.an('array');
});
pm.test("Cada produto tem a estrutura correta", function () {
    if (jsonData.quantidade > 0) {
        jsonData.produtos.forEach((produto, index) => {
            pm.expect(produto).to.have.property("nome");
            pm.expect(produto).to.have.property("preco");
            pm.expect(produto).to.have.property("descricao");
            pm.expect(produto).to.have.property("quantidade");
            pm.expect(produto).to.have.property("_id");
            pm.expect(produto.nome).to.be.a('string');
            pm.expect(produto.preco).to.be.a('number');
            pm.expect(produto.descricao).to.be.a('string');
            pm.expect(produto.quantidade).to.be.a('number');
            pm.expect(produto._id).to.be.a('string');
            pm.expect(produto.nome).to.not.be.empty;
            pm.expect(produto.preco).to.be.at.least(0);
            pm.expect(produto.quantidade).to.be.at.least(0);
            pm.expect(produto._id).to.not.be.empty;
        });
    }
});
pm.test("Quantidade corresponde ao número de produtos", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.quantidade).to.eql(jsonData.produtos.length);
});
pm.test("Response time é aceitável", function () {
    pm.expect(pm.response.responseTime).to.be.below(3000);
});

pm.test("Parâmetros de query são suportados", function () {
    pm.expect(pm.response.code).to.eql(200);
});
```

- `POST /produtos`

```
if (pm.response.code === 201) {
    pm.test("Status 201 - Cadastro realizado com sucesso", function () {
        pm.expect(pm.response.code).to.eql(201);
    });
    pm.test("Mensagem de cadastro bem-sucedido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
    });
    pm.test("ID do produto é retornado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("_id");
        pm.expect(jsonData._id).to.be.a('string');
        pm.expect(jsonData._id).to.not.be.empty;
        pm.environment.set("produto_id", jsonData._id);
        console.log("ID do produto criado:", jsonData._id);
    });
    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}

if (pm.response.code === 400) {
    pm.test("Status 400 - Nome duplicado", function () {
        pm.expect(pm.response.code).to.eql(400);
    });
    pm.test("Mensagem de nome já existente", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Já existe produto com esse nome");
    });
    pm.test("Resposta não contém ID em caso de erro", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.not.have.property("_id");
    });
}
if (pm.response.code === 401) {
    pm.test("Status 401 - Token inválido", function () {
        pm.expect(pm.response.code).to.eql(401);
    });
    pm.test("Mensagem de token inválido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Token de acesso ausente, inválido, expirado ou usuário do token não existe mais");
    });
}
if (pm.response.code === 403) {
    pm.test("Status 403 - Acesso negado", function () {
        pm.expect(pm.response.code).to.eql(403);
    });
    pm.test("Mensagem de rota exclusiva para administradores", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Rota exclusiva para administradores");
    });
}
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- `GET /produtos/{_id}`

```
if (pm.response.code === 200) {
    pm.test("Status 200 - Produto encontrado", function () {
        pm.expect(pm.response.code).to.eql(200);
    });
    pm.test("Resposta contém todos os campos do produto", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("nome");
        pm.expect(jsonData).to.have.property("preco");
        pm.expect(jsonData).to.have.property("descricao");
        pm.expect(jsonData).to.have.property("quantidade");
        pm.expect(jsonData).to.have.property("_id");
    });
    pm.test("Tipos de dados estão corretos", function () {
        var jsonData = pm.response.json();

        pm.expect(jsonData.nome).to.be.a('string');
        pm.expect(jsonData.preco).to.be.a('number');
        pm.expect(jsonData.descricao).to.be.a('string');
        pm.expect(jsonData.quantidade).to.be.a('number');
        pm.expect(jsonData._id).to.be.a('string');
    });
    pm.test("Valores são válidos", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData.nome).to.not.be.empty;
        pm.expect(jsonData.preco).to.be.at.least(0);
        pm.expect(jsonData.quantidade).to.be.at.least(0);
        pm.expect(jsonData._id).to.not.be.empty;
        const expectedId = pm.environment.get("produto_id") || "8eeJh5lz3k6kSIzA";
        pm.expect(jsonData._id).to.eql(expectedId);
    });
    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 200) {
    var jsonData = pm.response.json();
    if (jsonData.message === "Produto não encontrado") {
        pm.test("Produto não encontrado", function () {
            pm.expect(jsonData.message).to.eql("Produto não encontrado");
        });

        pm.test("Resposta de não encontrado não contém dados do produto", function () {
            pm.expect(jsonData).to.not.have.property("nome");
            pm.expect(jsonData).to.not.have.property("preco");
            pm.expect(jsonData).to.not.have.property("descricao");
            pm.expect(jsonData).to.not.have.property("quantidade");
        });
    }
}

pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- `DELETE /produtos/{_id}`

```
if (pm.response.code === 200) {
    pm.test("Status 200 - Operação concluída", function () {
        pm.expect(pm.response.code).to.eql(200);
    });
    pm.test("Resposta contém mensagem apropriada", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        const possibleMessages = [
            "Registro excluído com sucesso",
            "Nenhum registro excluído"
        ];
        pm.expect(possibleMessages).to.include(jsonData.message);
    });
    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 400) {
    pm.test("Status 400 - Não é permitido excluir produto com carrinho", function () {
        pm.expect(pm.response.code).to.eql(400);
    });
    pm.test("Mensagem de erro correta", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Não é permitido excluir produto que faz parte de carrinho");
    });
    pm.test("Resposta contém IDs dos carrinhos", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("carrinhos");
        pm.expect(jsonData.carrinhos).to.be.an('array');
        pm.expect(jsonData.carrinhos.length).to.be.at.least(1);
        pm.environment.set("carrinhos_bloqueio", JSON.stringify(jsonData.carrinhos));
        console.log("IDs dos carrinhos que impedem exclusão:", jsonData.carrinhos);
    });
}
if (pm.response.code === 401) {
    pm.test("Status 401 - Token inválido", function () {
        pm.expect(pm.response.code).to.eql(401);
    });
    pm.test("Mensagem de token inválido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Token de acesso ausente, inválido, expirado ou usuário do token não existe mais");
    });
}
if (pm.response.code === 403) {
    pm.test("Status 403 - Acesso negado", function () {
        pm.expect(pm.response.code).to.eql(403);
    });
    pm.test("Mensagem de rota exclusiva para administradores", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Rota exclusiva para administradores");
    });
}
if (pm.response.code === 404) {
    pm.test("Status 404 - Rota não encontrada", function () {
        pm.expect(pm.response.code).to.eql(404);
    });
}

pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

- `PUT /produtos/{_id}`

```
if (pm.response.code === 200) {
    pm.test("Status 200 - Alteração realizada com sucesso", function () {
        pm.expect(pm.response.code).to.eql(200);
    });
    pm.test("Mensagem de alteração bem-sucedida", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Registro alterado com sucesso");
    });
    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 201) {
    pm.test("Status 201 - Cadastro realizado com sucesso", function () {
        pm.expect(pm.response.code).to.eql(201);
    });
    pm.test("Mensagem de cadastro bem-sucedido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
    });
    pm.test("ID do novo produto é retornado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("_id");
        pm.expect(jsonData._id).to.be.a('string');
        pm.expect(jsonData._id).to.not.be.empty;
        pm.environment.set("novo_produto_id", jsonData._id);
        console.log("ID do novo produto criado:", jsonData._id);
    });
}
if (pm.response.code === 400) {
    pm.test("Status 400 - Nome duplicado", function () {
        pm.expect(pm.response.code).to.eql(400);
    });
    pm.test("Mensagem de nome já existente", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Já existe produto com esse nome");
    });
    pm.test("Resposta não contém ID em caso de erro", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.not.have.property("_id");
    });
}
if (pm.response.code === 401) {
    pm.test("Status 401 - Token inválido", function () {
        pm.expect(pm.response.code).to.eql(401);
    });
    pm.test("Mensagem de token inválido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Token de acesso ausente, inválido, expirado ou usuário do token não existe mais");
    });
}
if (pm.response.code === 403) {
    pm.test("Status 403 - Acesso negado", function () {
        pm.expect(pm.response.code).to.eql(403);
    });
    pm.test("Mensagem de rota exclusiva para administradores", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Rota exclusiva para administradores");
    });
}
```

- **Carrinhos**
  - `GET /carrinhos`

```
pm.test("Status 200 - Lista de carrinhos retornada", function () {
    pm.expect(pm.response.code).to.eql(200);
});
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});
pm.test("Estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
pm.test("Resposta contém estrutura correta", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("quantidade");
    pm.expect(jsonData.quantidade).to.be.a('number');
    pm.expect(jsonData.quantidade).to.be.at.least(0);
    pm.expect(jsonData).to.have.property("carrinhos");
    pm.expect(jsonData.carrinhos).to.be.an('array');
});

pm.test("Quantidade corresponde ao número de carrinhos", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.quantidade).to.eql(jsonData.carrinhos.length);
});
if (pm.response.json().quantidade > 0) {
    pm.test("Cada carrinho tem estrutura correta", function () {
        var jsonData = pm.response.json();
        jsonData.carrinhos.forEach((carrinho, index) => {
            pm.expect(carrinho).to.have.property("produtos");
            pm.expect(carrinho).to.have.property("precoTotal");
            pm.expect(carrinho).to.have.property("quantidadeTotal");
            pm.expect(carrinho).to.have.property("idUsuario");
            pm.expect(carrinho).to.have.property("_id");
            pm.expect(carrinho.produtos).to.be.an('array');
            pm.expect(carrinho.precoTotal).to.be.a('number');
            pm.expect(carrinho.quantidadeTotal).to.be.a('number');
            pm.expect(carrinho.idUsuario).to.be.a('string');
            pm.expect(carrinho._id).to.be.a('string');
            pm.expect(carrinho.precoTotal).to.be.at.least(0);
            pm.expect(carrinho.quantidadeTotal).to.be.at.least(0);
            pm.expect(carrinho.idUsuario).to.not.be.empty;
            pm.expect(carrinho._id).to.not.be.empty;
        });
    });
    pm.test("Produtos dentro do carrinho têm estrutura correta", function () {
        var jsonData = pm.response.json();

        jsonData.carrinhos.forEach((carrinho, carrinhoIndex) => {
            if (carrinho.produtos.length > 0) {
                carrinho.produtos.forEach((produto, produtoIndex) => {
                    pm.expect(produto).to.have.property("idProduto");
                    pm.expect(produto).to.have.property("quantidade");
                    pm.expect(produto).to.have.property("precoUnitario");
                    pm.expect(produto.idProduto).to.be.a('string');
                    pm.expect(produto.quantidade).to.be.a('number');
                    pm.expect(produto.precoUnitario).to.be.a('number');
                    pm.expect(produto.idProduto).to.not.be.empty;
                    pm.expect(produto.quantidade).to.be.at.least(1);
                    pm.expect(produto.precoUnitario).to.be.at.least(0);
                });
            }
        });
    });
}

pm.test("Response time é aceitável", function () {
    pm.expect(pm.response.responseTime).to.be.below(3000);
});
```

- `POST /carrinhos`

```
if (pm.response.code === 201) {
    pm.test("Status 201 - Cadastro realizado com sucesso", function () {
        pm.expect(pm.response.code).to.eql(201);
    });
    pm.test("Mensagem de cadastro bem-sucedido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
    });
    pm.test("ID do carrinho é retornado", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("_id");
        pm.expect(jsonData._id).to.be.a('string');
        pm.expect(jsonData._id).to.not.be.empty;
        pm.environment.set("carrinho_id", jsonData._id);
        console.log("ID do carrinho criado:", jsonData._id);
    });

    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 400) {
    pm.test("Status 400 - Algo deu errado", function () {
        pm.expect(pm.response.code).to.eql(400);
    });
    pm.test("Mensagem de erro específica", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        const possibleMessages = [
            "Não é permitido possuir produto duplicado",
            "Não é permitido ter mais de 1 carrinho",
            "Produto não encontrado",
            "Produto não possui quantidade suficiente"
        ];
        const messageFound = possibleMessages.some(msg => jsonData.message.includes(msg));
        pm.expect(messageFound).to.be.true;
    });

    pm.test("Resposta não contém ID em caso de erro", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.not.have.property("_id");
    });
}

// Teste para status 401 - Token inválido
if (pm.response.code === 401) {
    pm.test("Status 401 - Token inválido", function () {
        pm.expect(pm.response.code).to.eql(401);
    });

    pm.test("Mensagem de token inválido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Token de acesso ausente, inválido, expirado ou usuário do token não existe mais");
    });
}

// Testes que se aplicam a todos os casos
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- `GET /carrinhos/{_id}`

```
if (pm.response.code === 200) {
    const response = pm.response.json();
    if (response.message === "Carrinho não encontrado") {
        pm.test("Carrinho não encontrado", function () {
            pm.expect(response.message).to.eql("Carrinho não encontrado");
        });
        pm.test("Resposta de não encontrado não contém dados do carrinho", function () {
            pm.expect(response).to.not.have.property("produtos");
            pm.expect(response).to.not.have.property("precoTotal");
            pm.expect(response).to.not.have.property("quantidadeTotal");
            pm.expect(response).to.not.have.property("idUsuario");
            pm.expect(response).to.not.have.property("_id");
        });
    } else {
        pm.test("Status 200 - Carrinho encontrado", function () {
            pm.expect(pm.response.code).to.eql(200);
        });
        pm.test("Resposta contém todos os campos do carrinho", function () {
            pm.expect(response).to.have.property("produtos");
            pm.expect(response).to.have.property("precoTotal");
            pm.expect(response).to.have.property("quantidadeTotal");
            pm.expect(response).to.have.property("idUsuario");
            pm.expect(response).to.have.property("_id");
        });
        pm.test("Tipos de dados estão corretos", function () {
            pm.expect(response.produtos).to.be.an('array');
            pm.expect(response.precoTotal).to.be.a('number');
            pm.expect(response.quantidadeTotal).to.be.a('number');
            pm.expect(response.idUsuario).to.be.a('string');
            pm.expect(response._id).to.be.a('string');
        });
        pm.test("Valores são válidos", function () {
            pm.expect(response.idUsuario).to.not.be.empty;
            pm.expect(response._id).to.not.be.empty;
            const expectedId = pm.environment.get("carrinho_id") || "W6H6vGnDjUh6jkE1";
            pm.expect(response._id).to.eql(expectedId);
        });
        if (response.produtos.length > 0) {
            pm.test("Produtos dentro do carrinho têm estrutura correta", function () {
                response.produtos.forEach((produto, index) => {
                    pm.expect(produto).to.have.property("idProduto");
                    pm.expect(produto).to.have.property("quantidade");
                    pm.expect(produto).to.have.property("precoUnitario");

                    pm.expect(produto.idProduto).to.be.a('string');
                    pm.expect(produto.quantidade).to.be.a('number');
                    pm.expect(produto.precoUnitario).to.be.a('number');

                    pm.expect(produto.idProduto).to.not.be.empty;
                    pm.expect(produto.quantidade).to.be.at.least(1);
                    pm.expect(produto.precoUnitario).to.be.at.least(0);
                });
            });
        }
        pm.test("Response time é aceitável", function () {
            pm.expect(pm.response.responseTime).to.be.below(2000);
        });
    }


pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- `DELETE /carrinhos/concluir-compra`

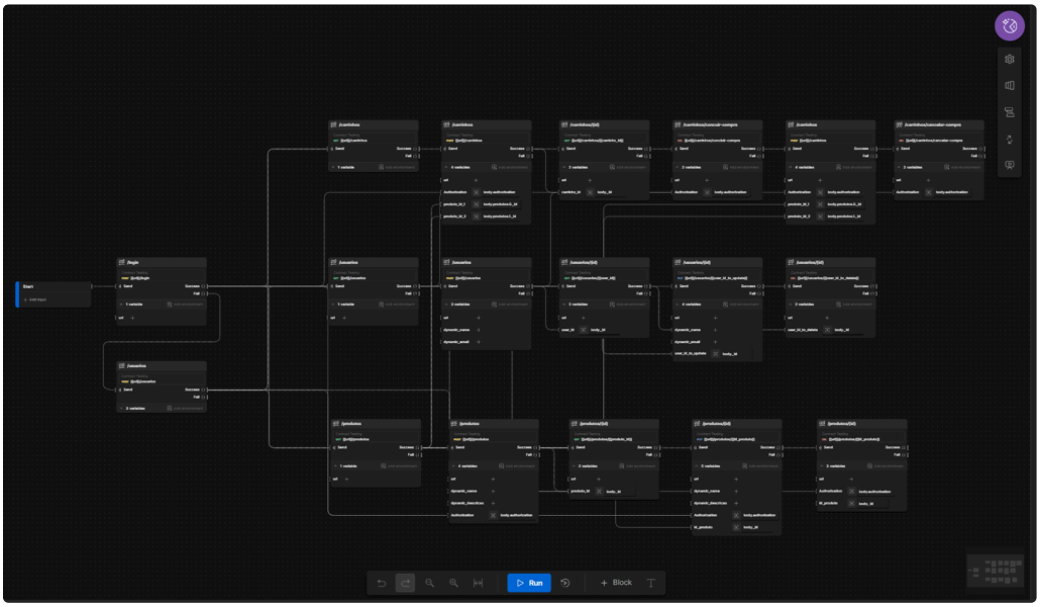
```
if (pm.response.code === 200) {
    pm.test("Status 200 - Operação concluída", function () {
        pm.expect(pm.response.code).to.eql(200);
    });

    pm.test("Resposta contém mensagem apropriada", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");

        // Pode retornar duas mensagens diferentes no status 200
        const possibleMessages = [
            "Registro excluído com sucesso",
            "Não foi encontrado carrinho para esse usuário"
        ];

        pm.expect(possibleMessages).to.include(jsonData.message);
    });

    pm.test("Response time é aceitável", function () {
        pm.expect(pm.response.responseTime).to.be.below(2000);
    });
}
if (pm.response.code === 401) {
    pm.test("Status 401 - Token inválido", function () {
        pm.expect(pm.response.code).to.eql(401);
    });

    pm.test("Mensagem de token inválido", function () {
        var jsonData = pm.response.json();
        pm.expect(jsonData).to.have.property("message");
        pm.expect(jsonData.message).to.eql("Token de acesso ausente, inválido, expirado ou usuário do token não existe mais");
    });
}
pm.test("Content-Type é application/json", function () {
    pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
});

pm.test("Resposta tem estrutura JSON válida", function () {
    pm.response.to.have.jsonBody();
});
```

- `DELETE /carrinhos/cancelar-compra`

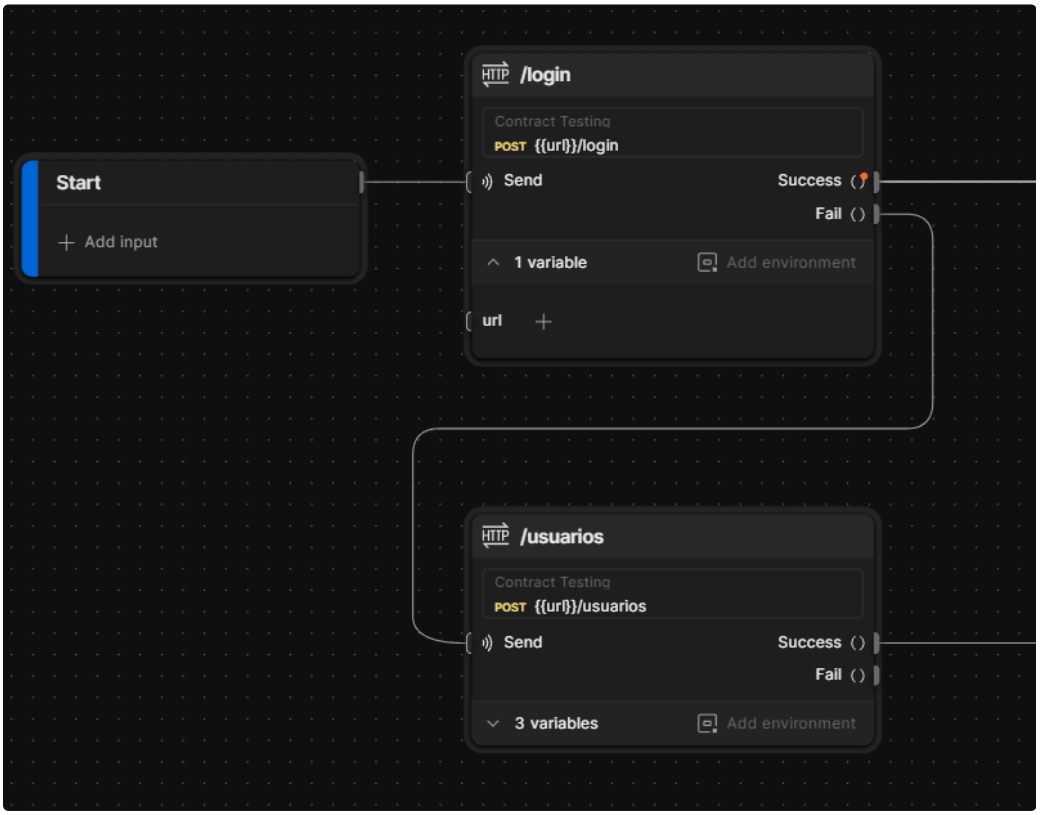
```
if (pm.response.code === 200) {
    pm.test("Status code is 200", function () {
        pm.response.to.have.status(200);
    });

    // Verificar se a resposta é JSON
    pm.test("Response is JSON", function () {
        pm.response.to.have.header("Content-Type", "application/json");
    });

    // Verificar a mensagem de sucesso
    pm.test("Success message in response", function () {
        const responseData = pm.response.json();
        pm.expect(responseData).to.have.property("message");
        pm.expect(responseData.message).to.be.oneOf([
            "Registro excluído com sucesso",
            "Não foi encontrado carrinho para esse usuário"
        ]);
    });
}

// Verificar se há erro de autenticação (401)
if (pm.response.code === 401) {
    pm.test("Status code is 401", function () {
        pm.response.to.have.status(401);
    });

    pm.test("Token error message", function () {
        const responseData = pm.response.json();
        pm.expect(responseData).to.have.property("message");
        pm.expect(responseData.message).to.include("Token de acesso ausente");
    });
}

// Verificar tempo de resposta
pm.test("Response time is acceptable", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000); // Menos de 1 segundo
});
```

Regressão

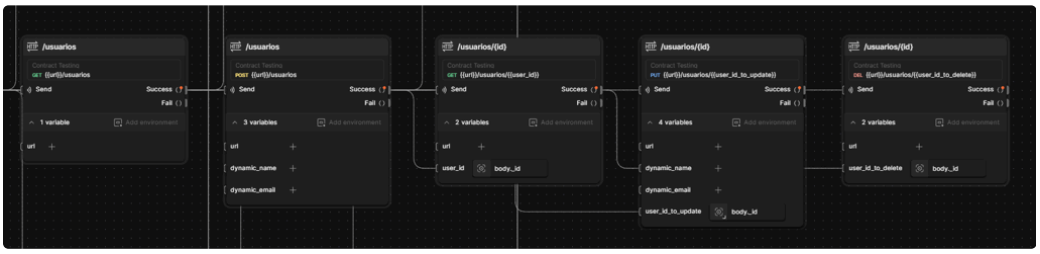
Todos os testes por rotas foram automatizados através das ferramentas propostas para a análise do fluxo geral do sistema no teste de regressão. O Teste automatizado possui o seguinte fluxo.



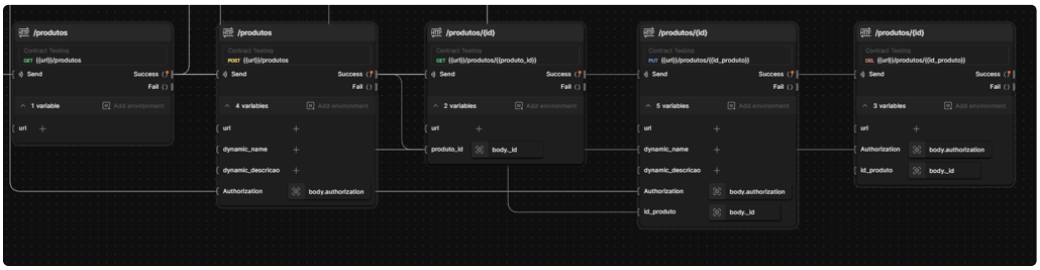
1. Criação de Um login com usuário para permissões de administrador



2. Fluxo dos usuários

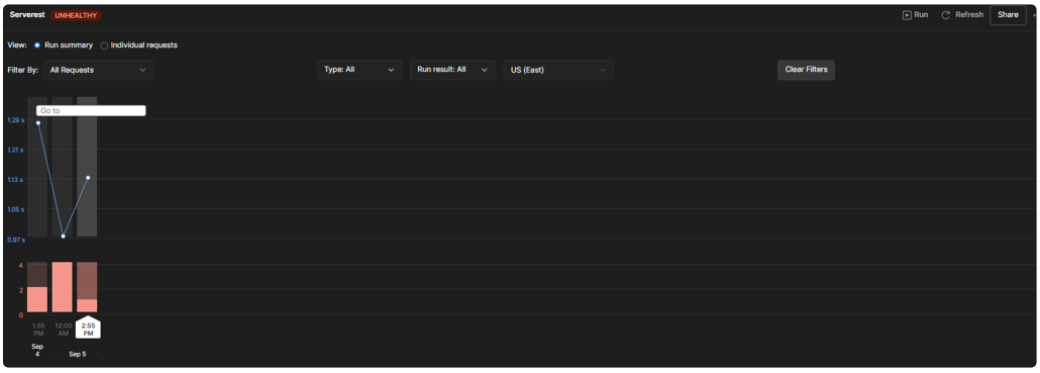


3. Fluxo dos Produtos

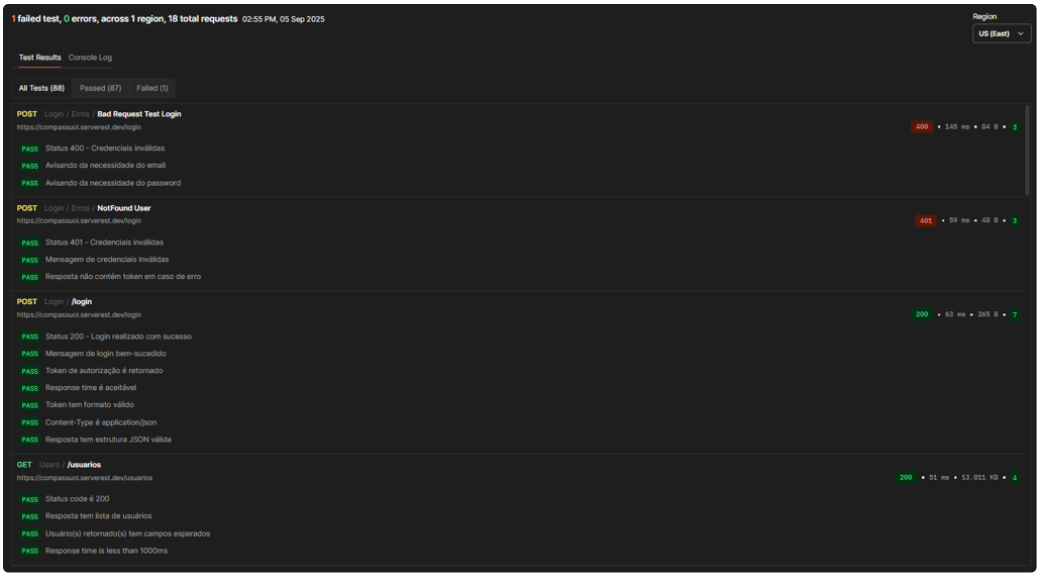


4.Fluxo do Carrinho

Todo o fluxo mostrado foi automatizado e pode ser executado com mais detalhes pelo Monitor no Postman.



Foi executados 26 vezes par um aprimoramento e teste da automação. Sendo possível verificar as Testes falhos e os testes passados pelo Run.



Testes no Robot Framework

TC01 - Login Válido

Endpoint: `POST /login`

Parâmetros de Entrada:

- `email` : fulano@qa.com
- `password` : teste

Status Esperado: 200

Validação: Verificação do status code 200

TC02 - Login Inválido

Endpoint: `POST /login`

Parâmetros de Entrada:

- `email` : invalido@test.com
- `password` : senha

Status Esperado: 401

Validação: Verificação do status code 401

TC03 - Listar Usuários

Endpoint: `GET /usuarios`

Parâmetros de Entrada: Nenhum

Status Esperado: 200

Validação: Verificação do status code 200

TC04 - Cadastrar Usuário

Endpoint: `POST /usuarios`

Parâmetros de Entrada:

- `nome` : Nome Teste
- `email` : [gerado dinamicamente]
- `password` : senha123
- `administrador` : true

Status Esperado: 201

Validação:

- Verificação do status code 201
- Verificação da presença do campo `_id` na resposta
- Cleanup: Deletar o usuário criado

TC05 - Cadastrar Usuário Email Duplicado

Endpoint: `POST /usuarios`

Parâmetros de Entrada:

- `nome` : Nome
- `email` : [mesmo email usado duas vezes]
- `password` : senha
- `administrador` : false

Status Esperado: 400 (segunda requisição)

Validação:

- Primeira requisição retorna 201
- Segunda requisição com mesmo email retorna 400
- Cleanup: Deletar o primeiro usuário criado

TC06 - Buscar Usuário Por ID

Endpoint: `GET /usuarios/{_id}`

Parâmetros de Entrada:

- `_id` : ID do usuário criado previamente

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Cadastrar usuário antes da busca
- Cleanup: Deletar o usuário após o teste

TC07 - Buscar Usuário ID Inexistente

Endpoint: `GET /usuarios/{_id}`

Parâmetros de Entrada:

- `_id` : IDinvalido123

Status Esperado: 400

Validação: Verificação do status code 400

TC08 - Atualizar Usuário

Endpoint: `PUT /usuarios/{_id}`

Parâmetros de Entrada:

- `_id` : ID do usuário existente
- `nome` : Novo Nome
- `email` : [novo email gerado]
- `password` : nova
- `administrador` : false

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Cadastrar usuário antes da atualização
- Cleanup: Deletar o usuário após o teste

TC09 - Deletar Usuário

Endpoint: `DELETE /usuarios/{_id}`

Parâmetros de Entrada:

- `_id` : ID do usuário existente

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Cadastrar usuário antes da exclusão

TC10 - Listar Produtos

Endpoint: `GET /produtos`

Parâmetros de Entrada: Nenhum

Status Esperado: 200

Validação: Verificação do status code 200

TC11 - Cadastrar Produto

Endpoint: `POST /produtos`

Parâmetros de Entrada:

- `nome` : Produto [string aleatória]
- `preco` : 100
- `descricao` : Desc
- `quantidade` : 50
- **Header:** `Authorization` : [token obtido via login admin]

Status Esperado: 201

Validação:

- Verificação do status code 201
- Verificação da presença do campo `_id`
- Setup: Login como admin (fulano@qa.com)
- Cleanup: Deletar o produto criado

TC12 - Cadastrar Produto Sem Token

Endpoint: `POST /produtos`

Parâmetros de Entrada:

- `nome` : Produto [string aleatória]
- `preco` : 100
- `descricao` : Desc
- `quantidade` : 50
- **Header:** `Authorization` : [vazio]

Status Esperado: 401

Validação: Verificação do status code 401

TC13 - Buscar Produto Por ID

Endpoint: `GET /produtos/{_id}`

Parâmetros de Entrada:

- `_id` : ID do produto criado previamente

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Login admin e cadastrar produto
- Cleanup: Deletar o produto

TC14 - Buscar Produto ID Inexistente

Endpoint: `GET /produtos/{_id}`

Parâmetros de Entrada:

- `_id` : IDinvalido123

Status Esperado: 400

Validação: Verificação do status code 400

TC15 - Atualizar Produto

Endpoint: `PUT /produtos/{_id}`

Parâmetros de Entrada:

- `_id` : ID do produto existente
- `nome` : Novo [string aleatória]
- `preco` : 200
- `descricao` : Nova
- `quantidade` : 25
- **Header:** `Authorization` : [token admin]

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Login admin e cadastrar produto
- Cleanup: Deletar o produto

TC16 - Deletar Produto

Endpoint: `DELETE /produtos/{_id}`

Parâmetros de Entrada:

- `_id` : ID do produto existente
- **Header:** `Authorization` : [token admin]

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Login admin e cadastrar produto

TC17 - Listar Carrinhos

Endpoint: `GET /carrinhos`

Parâmetros de Entrada: Nenhum

Status Esperado: 200

Validação: Verificação do status code 200

TC18 - Cadastrar Carrinho

Endpoint: `POST /carrinhos`

Parâmetros de Entrada:

- `produtos`: [{"idProduto": [ID válido], "quantidade": 2}]
- **Header:** `Authorization`: [token usuário comum]

Status Esperado: 201

Validação:

- Verificação do status code 201
- Setup: Login admin, criar produto, criar usuário comum, login como usuário
- Cleanup: Cancelar compra, deletar produto e usuário

TC19 - Buscar Carrinho Por ID

Endpoint: `GET /carrinhos/{_id}`

Parâmetros de Entrada:

- `_id`: ID do carrinho criado previamente

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Login admin, criar produto, criar usuário, login usuário, criar carrinho
- Cleanup: Cancelar compra, deletar produto e usuário

TC20 - Concluir Compra

Endpoint: `DELETE /carrinhos/concluir-compra`

Parâmetros de Entrada:

- **Header:** `Authorization`: [token usuário com carrinho ativo]

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Criar produto, usuário e carrinho
- Cleanup: Deletar produto e usuário

TC21 - Cancelar Compra

Endpoint: `DELETE /carrinhos/cancelar-compra`

Parâmetros de Entrada:

- **Header:** `Authorization` : [token usuário com carrinho ativo]

Status Esperado: 200

Validação:

- Verificação do status code 200
- Setup: Criar produto, usuário e carrinho
- Cleanup: Deletar produto e usuário

Resumo Estatístico

Total de Casos de Teste: 21

Distribuição por Módulo:

- Login: 2 testes
- Usuários: 7 testes
- Produtos: 7 testes
- Carrinhos: 5 testes